Escuela Politécnica Superior

Departamento de Ingeniería Informática

TESIS DOCTORAL

Modelo programable para la serialización y evaluación de modelos heterogéneos en clientes web

AUTOR:

JESÚS ENRIQUE CHAVARRIAGA BAUTISTA

DIRECTOR: **FERNANDO DÍEZ RUBIO**

CODIRECTOR: **FRANCISCO JURADO MONROY**

Madrid 2017

Título de tesis PhD: Modelo programable para la serialización y evaluación de modelos heterogéneos en clientes web

Autor: Jesús Enrique Chavarriaga Bautista

Afiliación: Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Madrid, España

Director: Fernando Díez Rubio

Codirector: Francisco Jurado Monroy

Resumen

Los Lenguajes de Dominio Específico (*Domain Specific Language*, DSL) son lenguajes de programación especializados que ofrecen abstracciones y técnicas de alto nivel para facilitar la comprensión y el funcionamiento de un dominio específico o parte de él. En el caso particular de los DSL definidos mediante gramáticas XML (XML-DSL), se pueden aprovechar las características propias de la especificación XML y de las APIs existentes en multitud de lenguajes de programación, permitiendo que resulten fáciles de manipular, sean extensibles además de combinables con otros lenguajes XML, pudiendo así usarse en diferentes plataformas y sistemas de información. En concreto, la presente Tesis Doctoral se centra en la creación y evaluación de XML-DSLs en clientes web para crear frameworks, componentes y/o aplicaciones web para dar soporte a soluciones de problemas específicos.

Las tecnologías web y la diversidad de dispositivos han cambiado el escenario para la creación de sistemas de información. Las aplicaciones web se basan en la creación de páginas web dinámicas, donde lenguajes XML como XSL, SVG y MathML enriquecen el modelo de presentación. La propuesta que se realiza en la presente Tesis Doctoral pretende facilitar la creación de lenguajes XML evaluables en clientes web. Para ello, se plantea un *motor de evaluación* que implementa un *intérprete* de XML-DSLs en JavaScript. Este intérprete tiene las facultades de evaluar múltiples programas para múltiples lenguajes en cualquier navegador y dispositivo, con capacidad de coexistencia e intercambio de información entre programas, y capacidad de administración de múltiples fuentes heterogéneas XML/JSON de forma nativa. Asimismo, para facilitar la creación e implementación de estos XML-DSLs se propone un *modelo de programación* y se implementa un *entorno de desarrollo ligero* sobre cliente web.

Más allá de las representaciones textuales, los DSL pueden tener una representación gráfica. Es así como surge el concepto de Lenguaje Visual de Dominio Específico (*Domain Specific Visual Language*, DSVL). Hasta el presente, la mayor parte de las propuestas de DSVLs están orientadas a resolver problemas en aplicaciones de escritorio y en Entornos de Desarrollo Integrado. Sin embargo, en clientes web la mayoría de aplicaciones y frameworks están encaminadas a la elaboración de diagramas y modelos gráficos, y a la visualización de información heterogénea en línea, es decir, no hay entornos de modelado visual o herramientas visuales para la creación y evaluación de DSVLs basadas en las tecnologías de un cliente web. En este marco de referencia, en la presente Tesis Doctoral se presenta el concepto de *diagrama programable* para definir librerías gráficas generadas a partir de SVG, y serializar modelos gráficos y DSVLs con gramática XML sobre una aplicación web. Este diagrama programable tiene la capacidad de asociar fuentes heterogéneas de información XML/JSON, añadir herramientas visuales, usar elementos de programación, asociar herramientas de generación automática de diagramas y proponer mecanismos de recomendación de diagramas. Además, gracias a que la serialización de los modelos gráficos se realiza como un XML-DSL, estos se pueden evaluar y ejecutar directamente en el entorno del cliente web.

Para validar las diferentes características y funcionalidades sobre un cliente web de las propuestas realizadas en la presente Tesis Doctoral, se siguió la metodología cualitativa de casos de estudio creando diversos escenarios. La transferencia tecnológica al sector industrial de las diferentes propuestas que se realizan en esta Tesis, ha permitido que la mayoría de estos escenarios hayan sido ampliamente probados y validados, aportando fiabilidad y robustez a las contribuciones presentadas.

Abstract

Domain Specific Language (DSL) are specialized programming languages that offer high level abstractions and techniques to facilitate the understanding and functionality of a specific domain or part of it. In the particular case of DSLs defined using XML grammars (XML-DSL), we can take advantage of the features of the XML specification and of the existing APIs in a lot of programming languages, allowing them to be easy to manipulate, extendable as well as combinable with other XML languages, and thus, being able to be used in different platforms and information systems. Specifically, this Doctoral Dissertation focuses on the creation and evaluation of XML-DSLs in web clients to create frameworks, components and/or web applications to provide solutions to specific problems.

Web technologies and device diversity have changed the scenario for the creation of information systems. Web applications are based on creating dynamic web pages, where XML languages like XSL, SVG and MathML enrich the presentation model. The proposal made in this Doctoral Dissertation aims to facilitate the creation of XML languages evaluable in web clients. To do this, we propose an *evaluation engine* that implements an XML-DSL *interpreter* in JavaScript. This interpreter has the ability to evaluate multiple programs for multiple languages in any browser and device, with the ability to coexist and exchange information between programs, and the ability to manage multiple heterogeneous XML/JSON data sources natively. Also, to facilitate the creation and implementation of these XML-DSLs, a *programming model* is proposed and a *lightweight development environment* on web is implemented.

Beyond textual representations, DSLs can have a graphical representation. This is how the concept of Domain Specific Visual Language (DSVL) arises. Currently, most DSVL proposals are geared toward solving problems in desktop applications and in Integrated Development Environments. However, in web clients most applications and frameworks are aimed at the development of diagrams and graphic models, and the visualization of heterogeneous information online, that is, there are no visual modelling environments or visual tools for the creation and evaluation of DSVLs based on the technologies of web clients. In this context, this Doctoral Dissertation presents the concept of a *programmable diagram* to define graphic libraries generated from SVG, and to serialize graphic models and DSVLs with XML grammar on a web application. This programmable diagram has the ability to associate heterogeneous data sources in XML/JSON, to add visual tools, to use programming elements, to associate automatic diagram generation tools and to propose diagram recommendation mechanisms. In addition, because the serialization of the graphic models is done as XML-DSL, they can be evaluated and executed directly in the web client environment.

To validate the different features and functionalities on a web client of the proposals made in this Doctoral Dissertation, the qualitative methodology of case studies was followed by creating different scenarios. The technological transfer to the industrial sector of the proposals made in this Dissertation has allowed us to widely test and validate most of these scenarios, contributing to the reliability and robustness of the presented contributions.

Agradecimientos

Esta tesis representa un hito muy importante en mis objetivos de vida. De niño soñaba con ser pintor, bailarín, jugador o dueño de una discoteca. La motivación de mi Madre por el estudio, el gusto por las matemáticas y las ganas de aprender siempre algo nuevo día a día, me llevaron a conseguir otro tipo de sueño, que sin dudarlo, es mucho más enriquecedor para el espíritu. Agradezco a todas las personas que han contribuido a alcanzar esta importante meta.

En primer lugar, a mis tutores Fernando y Paco que con su experiencia, conocimientos, consejos, apoyo y paciencia me ayudaron a aprender y crecer en este proceso.

A Francy que con su optimismo, paciencia y confianza ha sido el principal apoyo para este proyecto.

A B2T Concept por la oportunidad de trabajar, exponer ideas y facilitar la creación de diferentes soluciones a su Plataforma SCOOP.

A mi familia que desde lejos me ha animado, motivado y nunca ha dudado de lo importante que es para mí lograr esta meta.

A mis amigos en Colombia que me respaldaron confirmando que no importa el tiempo y la distancia.

A mis amigos en España que hicieron más fácil y agradable este período, fueron soporte en las dificultades y compañeros en los buenos momentos.

A todos muchas gracias.

Tabla de Contenido

Resumen v
Abstract vii
Agradecimientos ix
Tabla de Contenido xi
Lista de Figuras xv
Lista de Tablas xxi
Glosario de Acrónimos xxiii
Capítulo 1. Introducción 1
1.1 Motivación 1
1.2 Descripción del problema 2
1.2.1 Lenguajes de Dominio Específico 3
1.2.2 Lenguajes de Dominio Específico basados en gramáticas XML 3
1.2.3 Lenguajes Visuales de Dominio Específico 4
1.2.4 Trabajos previos de creación de DSLs Textuales, XML-DSLs y DSVLs 5
1.2.5 MDE y Plataforma B2T SCOOP 6
1.3 Objetivos de investigación 7
1.4 Resultados 9
1.4.1 Transferencia Tecnológica 9
1.4.2 Publicaciones 10
1.4.3 Resumen 12
1.5 Estructura de la tesis 13
Capítulo 2. Estado del Arte 15
2.1 Conceptos Previos 15
2.1.1 Entorno Cliente Web (CSWE) y Aplicaciones Web 15
2.1.2 La Web 2.0 17
2.1.3 Los intérpretes de programación 17
2.1.4 Patrones y Desarrollo basado en Componentes 18
2.2 Los Lenguajes 19
2.2.1 Plataforma Web Abierta y los lenguajes XML 19
2.2.2 Lenguajes de Dominio Específico 20
2.2.3 Lenguajes Visuales 21
2.3 Frameworks y Entornos de Desarrollo Integrado para la creación de DSLs y DVSLs 21
2.3.1 Frameworks para la creación de DSLs 22
2.3.2 Herramientas para la creación de Diagramas y Lenguajes Visuales en CSWE 23
2.4 Identificación de Funcionalidades para la creación de diagramas y lenguajes visuales 23
2.5 Mapeo Sistemático sobre DSLs en CSWE 25
2.5.1 Revisiones previas 25
2.5.2 Mapeo Sistemático 26
2.6 Resumen 34

Capítulo 3. PsiEngine: Motor de Evaluación Psi 37
3.1 Introducción 37
3.2 Conceptos de Programación JavaScript 39
3.2.1 Elementos básicos de programación en JavaScript 39
3.2.2 Implementación del concepto Clase 39
3.2.3 Implementación del concepto de Componente 41
3.3 El Lenguaje Psi 41
3.3.1 Formalización de la gramática Psi 42
3.3.2 Formalización del lenguaje Psi y el componente Psi 43
3.4 Metodología para el diseño e implementación de lenguajes Psi 45
3.5 El Intérprete PsiXML 46
3.5.1 Formalización del Intérprete 47
3.5.2 Implementación del Intérprete 48
3.6 Implementación del PsiEngine 50
3.7 Caso de estudio: Lenguaje MiniPsi 51
3.7.1 Análisis y diseño del Lenguaje MiniPsi 51
3.7.2 Diseño del Lenguaje MiniPsi y del Componente MiniPsi 51
3.7.3 Implementación del Componente MiniPsi 52
3.7.4 Ejemplos de uso del lenguaje MiniPsi 53
3.7.5 Aplicación Web “MiniPsi Language” 54
3.8 Resumen 55
Capítulo 4. PsiModel: Modelo de programación Psi para la generación de código JavaScript 57
4.1 Introducción 57
4.2 Formalización del PsiModel 59
4.3 Diseño de los lenguajes Psi para PsiModel 60
4.3.1 Definición de lenguajes 60
4.3.2 Ejemplo ilustrativo 62
4.3.3 Fuentes de generación Psi 63
4.4 Diseño de los componentes Psi de PsiModel 67
4.4.1 El Componente MPsi 68
4.4.2 El Componente MIPsi 69
4.5 Implementación de los componentes Psi de PsiModel 71
4.6 Resumen 72
Capítulo 5. PsiDiagram: Diagrama Programable Psi 73
5.1 Introducción 73
5.2 Creación y funcionalidad de un diagrama programable 75
5.3 Formalización de PsiDiagram 76
5.4 Metodología de Diseño para PsiDiagram 78
5.5 El Lenguaje GLPsi y el Lenguaje DPsi 78
5.5.1 Análisis y Diseño de los Lenguajes GLPsi y DPsi 79
5.5.2 Diseño del Componente GLPsi 84
5.5.3 Implementación del Componente GLPsi 86
5.5.4 Diseño del Componente DPsi 86
5.5.5 Implementación del Componente DPsi 88
5.6 El Lenguaje TPsi 89
5.6.1 Diseño de las Herramientas Visuales 89
5.6.2 Diseño de las Utilidades de Programación 93
5.6.3 Diseño del Componente TPsi 94
5.7 El Generador de Diagramas 96
5.7.1 Análisis y Diseño del Lenguaje Generator para TPsi 96
5.7.2 Diseño del Generator para TPsi 97
5.8 El Recomendador de Diagramas 99

5.8.1 Análisis y Diseño del Lenguaje Generator para TPsi 99
5.8.2 Diseño de Generator para TPsi 100
5.9 Implementación del Componente TPsi 101
5.10 Resumen 101
Capítulo 6. PsiEnvironment: Entorno de Desarrollo Psi 103
6.1 Introducción 103
6.2 Entorno de desarrollo 104
6.2.1 Menú Principal 104
6.2.2 Definición de Proyecto 105
6.2.3 Tipos de Proyecto 107
6.3 Métricas software para Proyectos de PsiEnvironment 108
6.3.1 Métricas software para código JavaScript 109
6.3.2 Definición de métricas para proyectos PsiModel 111
6.3.3 Definición de métricas para proyectos PsiDiagram 112
6.4 Generación de métricas software en PsiEnvironment 113
6.4.1 Proyectos PsiModel 113
6.4.2 Proyectos PsiDiagram 115
6.5 Resumen 116
Capítulo 7. Transferencia Tecnológica 119
7.1 Introducción: Plataforma B2T SCOOP 119
7.2 Proyecto ADG 120
7.2.1 Estructura del Proyecto 121
7.2.2 Tipos de diagramas ADG 122
7.3 Proyecto SID 125
7.3.1 Estructura del Proyecto 125
7.3.2 Tipos de diagrama SID y Aplicación SIDTest 127
7.4 Proyecto de BPME 129
7.4.1 Estructura del Proyecto 129
7.4.2 Visor de Proyectos BPMEs 131
7.4.3 Generación automática de diagramas BPM para BPMs y Eventos 131
7.5 Proyecto SMAPPS 132
7.5.1 Funcionamiento de una aplicación móvil basada en Servicios SCOOP 132
7.5.2 Servicios SMAPPS 133
7.5.3 Aplicación móvil basada en Servicios SMAPPS 135
7.5.4 El Lenguaje BPMEPsi y los Componentes AngularSMAPPS 137
7.6 Resumen 139
Capítulo 8. Metodología de Validación y Resultados 141
8.1 Metodología de Validación 141
8.2 Validación de objetivos PsiEngine y PsiModel 143
8.2.1 Casos de Estudio Básicos 145
8.2.2 Kernel: Proyecto PsiEngine 153
8.2.3 Kernel: Proyecto PsiModel 154
8.2.4 Componentes de PsiDiagram: Proyectos GLPsi, DPsi, TPsi y WPPsi 156
8.2.5 Componentes Móviles: Proyectos SMAPPS 158
8.2.6 Notas finales de validación 162
8.3 Validación de objetivos de PsiDiagram 163
8.3.1 Diagramas de Clases y Diagramas PsiLSD 164
8.3.2 Proyecto ADG: Generación Automática de Diagramas 168
8.3.3 Proyecto SID: SCOOP Interface Diagram 171
8.3.4 Proyecto BPME y SMAPPS 174
8.3.5 Notas finales de validación 176
8.4 Resumen de Resultados 177

Capítulo 9. Conclusiones y Trabajos Futuros 179
9.1 Introducción 179
9.2 Contribuciones 180
9.3 Grado de cumplimiento de Hipótesis 183
9.4 Trabajos Futuros 185
9.4.1 Consolidación de proyectos actuales 185
9.4.2 Nuevas Propuestas 186
9.5 Comentarios Finales 187
Referencias Bibliográficas 189
Anexo A. Generación de Código Psi 201
A.1. Fuentes de generación Psi del tipo enumerativo JavaScript 201
A.2. Fuentes de generación Psi de un objeto JavaScript 201
A.3. Fuentes de generación Psi de un Web Widget 202
Anexo B. Especificaciones Psi 205
B.1. Especificación GLPsi/DPsi de una Figura Psi 205
B.2. Especificación GLPsi/DPsi de una Lista Psi 207
B.3. Especificación GLPsi/DPsi de un Contenedor Psi 209
B.4. Especificación GLPsi de las etiquetas IconSVG, Settings y Events 211
B.5. Especificación GLPsi de las etiquetas Function, Script, Make y Defs 212
B.6. Especificación DPsi de los Utilidades de definición 213
B.7. Especificación DPsi de los Elementos Gráficos 215
B.8. Especificación TPsi de elementos de Contenido, Posición, Eventos y Opciones 216
B.9. Especificación TPsi de las Herramientas Visuales 217
B.10. Especificación TPsi de las Utilidades de Programación 219
B.11. Especificación TPsi del Generador de Diagrama 220
Anexo C. Especificaciones PsiGVA 223
C.1. Especificación PsiGVA para la gramática GLPsi 223
C.2. Especificación PsiGVA para la gramática DPsi 224
C.3. Especificación PsiGVA para la gramática TPsi 224
C.4. Especificación PsiGVA para la gramática BPMEPsi 226
Anexo D. Marcos de Página Web Psi 227
D.1. Definición Conceptual 227
D.2. Especificación del Lenguaje WPPsi 228
D.3. Componente WPPsi 230
Anexo E. Casos de Estudio Psi 233
E.1. Aplicación Web FeedPsi 233
E.2. Animación Anisha 236
E.3. Diagramación Grapher 238

Lista de Figuras

Figura 1-1. Distribución de las contribuciones de objetivos de investigación respecto a la importancia del aporte y su relevancia. 12
Figura 2-1. Arquitectura multinivel para la creación de aplicaciones web. 16
Figura 2-2. Intérprete de un *Programa* P escrito en un *Lenguaje Fuente* P/LF. 18
Figura 2-3. Distribución por tipo de publicación (revista, conferencias/congresos y trabajos previos), por fuente de distribución y publicaciones por año que mencionan DSLs o temas relacionados. 30
Figura 2-4. Número de publicaciones candidatas (color azul) y otros estudios (color rojos). Distribución de publicaciones candidatas por tipo de DSL. 30
Figura 2-5. XML-DSLs y/o DSLs Textuales por tipo de publicación (revista, conferencias/congresos y trabajos previos) y por fuente de distribución para las Publicaciones XML-DSLs. 31
Figura 2-6. IDEs para la creación de DSLs y formas de ejecución de las Publicaciones XML-DSLs. 32
Figura 2-7. DSVLs y/o DSLs Textuales y/o XML-DSLs por tipo de publicación (revista, conferencias/congresos y trabajos previos) y por fuente de distribución para las Publicaciones DSVLs. 33
Figura 2-8. IDEs para la creación de DSVLs y formas de ejecución de las Publicaciones DSVLs. 34
Figura 2-9. Resumen del Estado del Arte. 34
Figura 3-1. Motor de Interpretación y Evaluación de Soluciones Programables o Motor de Evaluación Psi (PsiEngine, Programmable Solutions Interpreter Engine). 38
Figura 3-2. PsiEngine: Definición de Lenguaje Psi, Gramática Psi y Componente Psi 42
Figura 3-3. (a) Definición del diagrama de estructura del lenguaje Psi, denotado por PsiLSD. (b) El validador de atributos de la gramática Psi, denotada por PsiGVA. 43
Figura 3-4. Definición de Lenguaje Psi $\mathbb{L}$, Gramática Psi $\mathbb{G}$, Componente Psi $\mathbb{K}$ y asociaciones entre $\mathbb{T} \leftrightarrow \mathbb{C}$. 44
Figura 3-5. PsiEngine: Diagrama de bloques del *Intérprete PsiXML* 47
Figura 3-6**.** Diagrama de estados para un programa Psi. 48
Figura 3-7. Diagrama de clases del *Intérprete PsiXML* (Componente PsiXML) 49
Figura 3-8. Documentación del API del Motor de Evaluación Psi (PsiEngine). 50
Figura 3-9. Diagrama de clases y componentes de Motor de Evaluación Psi (PsiEngine). 51
Figura 3-10. (a) PsiLSD del lenguaje MiniPsi. (b) PsiGVA del lenguaje MiniPsi. (c) Diagrama de clases para el componente MiniPsi. 52
Figura 3-11. Aplicación Web MiniPsi: ejemplos de programación en lenguaje MiniPsi. 55
Figura 3-12. Diagrama de bloques para la creación de soluciones Psi basados en PsiEngine en un CSWE.56
Figura 3-13. Diagrama de bloques de la aplicación web “MiniPsi Language”. 56
Figura 4-1. Ejemplo de especificación e implementación de una función para el Modelo de Programación Psi (ModelPsi). 58
Figura 4-2. Esquema de PsiModel para la generación de código JavaScript y documentación. 59
Figura 4-3. Lenguajes MPsi y MIPsi del Modelo de Programación Psi (PsiModel). 60
Figura 4-4. Ejemplo de fuentes de generación Psi para una variable “var” de JavaScript. 62
Figura 4-5. Fuentes de generación Psi para una clase “class” de JavaScript. 63
Figura 4-6. Especificación MPsi e Implementación MIPsi para un Componente. 64

Figura 4-7. Plantilla XML para un Componente. 65
Figura 4-8. Especificación MPsi e Implementación MIPsi para un Componente Psi. 66
Figura 4-9. Plantilla XML para un Componente Psi. 67
Figura 4-10. Diagrama de Estructura PsiLSD del lenguaje MPsi. 68
Figura 4-11. Diagrama de Clases del Componente MPsi. 68
Figura 4-12. Diagrama de Estructura PsiLSD del Lenguaje MIPsi. 70
Figura 4-13. Diagrama de Clases del Componente MIPsi. 70
Figura 4-14. Documentación del API del Modelo de Generación Psi (PsiModel). 71
Figura 4-15. Diagrama de bloques para la creación de componentes JavaScript, widgets y/o componentes Psi, basados en PsiModel. 72
Figura 5-1. Definición de bloques de un *diagrama programable*. 74
Figura 5-2. Creación y visualización de un diagrama programable. Definición de *Librería Gráfica Psi* (GLPsi), *Diagramación Psi* (DPsi), *Herramientas Psi* (TPsi) y *Fuentes de Información* (DS). 76
Figura 5-3. Conjunto de componentes Psi del Diagrama Programable Psi. 77
Figura 5-4. Representación del proceso de *definición conceptual*, *representación, especificación Psi* e *implementación Psi* de un lenguaje Psi. 78
Figura 5-5. Definición conceptual de Línea Psi. 79
Figura 5-6. Ejemplo de implementación de la representación de Línea en SVG. 80
Figura 5-7. Especificación Psi de Línea Psi en el lenguaje GLPsi y en el lenguaje DPsi. 81
Figura 5-8. Definición conceptual de Figura. 82
Figura 5-9. Ejemplo de implementación de la definición de Figura en SVG. 83
Figura 5-10. Definición conceptual de Lista. 83
Figura 5-11. Definición conceptual de Contenedor. 84
Figura 5-12. Diagrama de Estructura del Lenguaje GLPsi. 85
Figura 5-13. Diagrama de clases del Componente GLPsi. 85
Figura 5-14. Diagrama de Estructura del Lenguaje DPsi (Utilidades de Configuación). 86
Figura 5-15. Diagrama de Estructura del Lenguaje DPsi (Elementos Gráficos). 87
Figura 5-16. Diagrama de clases del Componente DPsi. 88
Figura 5-17. Definiciones básicas de una Herramienta Visual. 89
Figura 5-18. Definición conceptual, representación HTML y especificación TPsi de la herramienta Menú Flotante. 90
Figura 5-19. Definición conceptual, representación HTML y especificación TPsi de la herramienta Menú. 90
Figura 5-20. Definición conceptual, representación HTML y especificación TPsi de la herramienta Ventana Emergente. 91
Figura 5-21. Definición conceptual, representación HTML y especificación TPsi de la herramienta Caja de Diálogo. 91
Figura 5-22. Definición conceptual, representación HTML y especificación TPsi de la herramienta Marco de Contenido. 92
Figura 5-23. Definición conceptual, representación HTML y especificación TPsi de la herramienta Barra de Herramientas. 92
Figura 5-24. Definición conceptual, representación HTML y especificación TPsi de la herramienta Soltar. 93
Figura 5-25. Definición conceptual, representación HTML y especificación TPsi de la herramienta Arrastras/Soltar. 93
Figura 5-26. Diagrama de Estructura del Herramientas Visuales de TPsi. 94
Figura 5-27. Diagrama de Estructura de Utilidades de Programación de TPsi. 95
Figura 5-28. Diagrama de clases de los principales elementos de programación del Componente TPsi. 95

Figura 5-29. Diagrama de clases del Componente TPsi para las Herramientas Visuales y Utilidades de Programación. 96
Figura 5-30. Definición conceptual (diagrama de proceso) del Generador de Diagramas. 97
Figura 5-31. Diagrama de Estructura del Generador de Diagramas de TPsi. 98
Figura 5-32. Diagrama de clases del Subcomponente Generator del Componente TPsi. 99
Figura 5-33. Definición conceptual (diagrama de proceso) del Recomendador de Diagramas. 99
Figura 5-34. Diagrama de Estructura del Recomendador de Diagramas de TPsi. 100
Figura 5-35. Diagrama de clases del Subcomponente Recommender del Componente TPsi. 101
Figura 5-36. Diagrama de bloques para la creación de diagramas, diagramas automáticos y/o DSVLs en un CSWE, basados en PsiDiagram. 102
Figura 6-1. Diagrama de bloques de la aplicación web PsiEnvironment. 104
Figura 6-2. Entorno de desarrollo PsiEnvironment. 105
Figura 6-3. Diagrama de bloques de un Proyecto Genérico. 107
Figura 6-4. Diagrama de bloques de un Proyecto PsiModel. 108
Figura 6-5. Diagrama de bloques de un proyecto de PsiDiagram. 108
Figura 6-6. Métricas en línea para proyectos PsiModel en PsiEnvironment. 114
Figura 6-7. Métricas en línea para proyectos PsiDiagram en PsiEnvironment. 116
Figura 6-8. Diagrama de bloques del Entorno de Desarrollo PsiEnvironment. 117
Figura 7-1. Plataforma B2T SCOOP. Lista de módulos que usan proyectos PsiModel y PsiDiagram. 120
Figura 7-2. Diagrama de bloques del Proyecto ADG. 121
Figura 7-3. Ejemplo de Diagrama de GANTT usando ADG. 122
Figura 7-4. Ejemplo de Modelo de Objetos SCOOP usando ADG. 122
Figura 7-5. Ejemplo de Diagrama WBT usando ADG. 123
Figura 7-6. Ejemplo de Diagrama Modelo de Datos SCOOP usando ADG. 123
Figura 7-7. Ejemplo de Diagrama de Estados usando ADG. 124
Figura 7-8. Ejemplo de Llamado de Funciones SCOOP usando ADG. 124
Figura 7-9. Ejemplo de Diagrama de Conceptos-Relaciones usando ADG. 124
Figura 7-10. Ejemplo de Diagrama PERT usando ADG. 125
Figura 7-11. Diagrama de bloques del Proyecto SID. 126
Figura 7-12. Ejemplo de Modelo de Conceptos-Relaciones de *Sketch* usando SID. 127
Figura 7-13. Ejemplo de Modelo de Estados de *Sketch* usando SID. 127
Figura 7-14. Ejemplo de Modelo de Procesos de *Sketch* usando SID. 128
Figura 7-15. Ejemplo de Modelo de Conceptos-Relaciones de *Modelo Técnico* usando SID. 128
Figura 7-16. Ejemplo de Modelo de Estados de *Modelo Técnico* usando SID. 128
Figura 7-17. Ejemplo de BPM de *BPMs y Eventos* usando SID. 129
Figura 7-18. Diagrama de bloques de proyectos BPME. 130
Figura 7-19. Ejemplo del Visor de Proyectos BPME. 131
Figura 7-20. Diagrama de creación de una aplicación móvil basada en servicios SMAPPS de B2T SCOOP. 133
Figura 7-21. Diagrama de bloques para la creación de Servicios SMAPPS. 134
Figura 7-22. Editor de diagramas BPMEBase para la definición de Servicio SMAPPS de SCOOP. 135
Figura 7-23. Diagrama de bloques de una Aplicación Móvil SMAPPS. 136
Figura 7-24. El PsiLSD del Lenguaje BPMEPsi. 137
Figura 7-25. Diagrama de clases para el componente BPMEPsi y su implementación en un Servicio SMAPPS. 137
Figura 7-26. Diagrama de clases para renderizar una página web de un servicio SMAPPS. 139
Figura 8-1. (a) Concisión de proyectos con Componentes Psi. (b) Concisión de proyectos con Componentes JavaScript. 145

Figura 8-2. Aplicación Web FeedPsi. Agregador de noticas al estilo Psi. 146
Figura 8-3. Ejemplo ilustrativo de Anisha. 148
Figura 8-4. Creación y visualización de figuras y líneas basadas en el concepto de definición gráfica.... 149
Figura 8-5. *Grapher*: entorno de desarrollo ligero para el uso de los componentes Graphs y Paint....... 150
Figura 8-6. Resumen de métricas para los Casos de Estudio Básicos. 151
Figura 8-7. Resumen de métricas del proyecto de componentes PsiEngine. 153
Figura 8-8. Fragmento de las métricas detalladas de los elementos de programación de PsiEngine. 154
Figura 8-9. Resumen de métricas del proyecto de componentes PsiModel. 155
Figura 8-10. Resumen de métricas para los proyectos de PsiDiagram. 157
Figura 8-11. Resumen de métricas para los proyectos AngularSMAPPS. 160
Figura 8-12 Métricas comparativas de los proyectos BPMEPsi, SMAPPS y AngularSMAPPS. 161
Figura 8-13. Líneas de código Psi escritas en GLPs, DPsi, TPsi, WPPs y DFPsi para PsiDiagram. 164
Figura 8-14. Editor de diagrama de clases y diagramas PsiLSD basado en el proyecto PsiBase de PsiDiagram. 165
Figura 8-15. Resumen de métricas del proyecto PsiBase. 166
Figura 8-16. Elementos gráficos de la diagramación Psi de los proyectos PsiClassDiagram, PsiClassBasic y PsiLSD. 167
Figura 8-17. Resumen de métricas del proyecto ADGBase. 169
Figura 8-18. Resumen de métricas PsiDiagram para los proyectos de los tipos de diagrama ADG. 170
Figura 8-19. Resumen de métricas del proyecto SIDBase. 172
Figura 8-20. Resumen de métricas PsiDiagram para los proyectos de los tipos de diagrama SID. 173
Figura 8-21. Resumen de métricas del proyecto BPMEBase. 175
Figura 8-22. Resumen de resultados para la validación de objetivos O1, O2, O3 y O4. 177
Figura 9-1. Diagrama visual de la estructura y contenido de la tesis. 181

Figura A-1. Fuentes de generación Psi para un tipo enumerativo de JavaScript. 201
Figura A-2. Fuentes de generación Psi para un objeto de JavaScript. 202
Figura A-3. Fuentes de generación Psi para un widget jQuery. 203

Figura B-1. Especificación de Figura Psi en el lenguaje GLPsi y en el lenguaje DPsi. 206
Figura B-2. Ejemplo de implementación de la representación de Lista en SVG. 207
Figura B-3. Especificación de Lista Psi en el lenguaje GLPsi y en el lenguaje DPsi. 208
Figura B-4. Ejemplo de implementación de la representación de Contenedor en SVG. 209
Figura B-5. Especificación de Contenedor Psi en el lenguaje GLPsi y en el lenguaje DPsi. 210

Figura D-1. Plantilla de diseño por secciones de una página web. 228
Figura D-2. (a) El PsiLSD para el lenguaje WPPsi. (b) El PsiGVA de la Gramática WPPsi. 228
Figura D-3. Diagrama de clases del Componente WPPsi. 230

Figura E-1. Diseño de una aplicación Web para la visualización de noticas a través de fuentes RSS, mediante la creación de un Lenguaje Psi. 234
Figura E-2. (a) PsiLSD para el lenguaje FeedPsi. (b) PsiGVA para el lenguaje FeedPsi. (c) Diagrama de clases para el Componente Feed. 234
Figura E-3. (a) PsiLSD para el lenguaje ShapesPsi. (b) PsiGVA para el lenguaje ShapesPsi. (c) Diagrama de clases para el Componente Shapes. 236
Figura E-4. (a) PsiLSD para el lenguaje AnimePsi. (b) PsiGVA para el lenguaje AnimePsi. (c) Diagrama de clases para el Componente Anime. 237

Figura E-5. Creación y visualización de figuras y líneas basadas en el concepto de definición gráfica. ...239
Figura E-6. (a) PsiLSD para el lenguaje GraphsPsi. (b) PsiGVA para la gramática GraphsPsi. (c) Diagrama de clases del componente Graphs. 240
Figura E-7. (a) PsiLSD para el lenguaje PaintPsi. (b) PsiGVA para la gramática PaintPsi. (c) Diagrama de clases del componente Paint. 242
Figura E-8. Ejemplo ilustrativo del encadenamiento de los lenguajes GraphsPsi y PaintPsi, con asociación de información XML. 244

Lista de Tablas

Tabla 1-1. Resumen de trabajos previos de creación DSLs textuales, XML-DSLs, DSVLs. 6
Tabla 2-1. Funcionalidades para la gestión de diagramas, asociación de herramientas visuales y asociación de información heterogénea. 24
Tabla 2-2. Análisis comparativo de herramientas para la gestión de modelos. 24
Tabla 2-3. Lista de resultados para el SMS con la cadena de búsqueda CB. 27
Tabla 2-4. Lista de publicaciones Candidatas del SMS. 27
Tabla 3-1. Etiquetas y clases asociadas del lenguaje MiniPsi. 52
Tabla 4-1. Atributos básicos de un elemento de programación Psi (Etiquetas $\mathbb{T}EPPsi$). 61
Tabla 4-2. Etiquetas y clases asociadas del lenguaje MPsi. 69
Tabla 4-3. Etiquetas y clases asociadas del lenguaje MIPsi. 70
Tabla 5-1. Implementación y documentación del componente GLPsi 86
Tabla 5-2. Implementación y documentación del componente DPsi 88
Tabla 5-3. Implementación y documentación del componente TPsi 101
Tabla 6-1. Lista de métricas software para código JavaScript. 109
Tabla 6-2. Lista de métricas software para los elementos de programación Psi. 112
Tabla 6-3. Lista de métricas para PsiDiagram. 112
Tabla 7-1. Proyectos Psi disponibles para el Proyecto GDA. 121
Tabla 7-2. Proyectos Psi disponibles para el Proyecto SID. 126
Tabla 7-3. Proyectos Psi disponibles para el Proyecto BPME 130
Tabla 7-4. Proyectos Psi disponibles para el Proyecto Servicio SMAPPS 134
Tabla 7-5. Diagramas BPME para la Aplicación Móvil B2TConcept 135
Tabla 7-6. Proyectos Psi disponibles para las Aplicaciones Móviles SMAPPS. 136
Tabla 7-7. Etiquetas y clases asociadas del lenguaje BPMEPsi 138
Tabla 8-1. Validación del Objetivo General VOG: Resumen de características/funcionalidades a validar en el objetivo general de esta tesis agrupados en PsiEngine, PsiModel y PsiDiagram. 142
Tabla 8-2. Ejemplo de validación de características del objetivo general. 143
Tabla 8-3. Lista de proyectos PsiModel para la creación de componentes y componentes Psi. 144
Tabla 8-4. Lista de proyectos Genéricos de los Casos de Estudio Básicos en PsiEnvironment. 149
Tabla 8-5. Proyectos FeedPsi, MiniPsi, Anisha y Grapher de tipo PsiModel en PsiEnvironment. 150
Tabla 8-6. Métricas para los componentes y componentes Psi de los Casos de Estudio Básicos. 152
Tabla 8-7. VOG: Resumen de características validadas en los Casos de Estudio Básicos para PsiEngine y PsiModel. 152
Tabla 8-8. Proyecto PsiEngine en PsiEnvironment. 153
Tabla 8-9. Resumen VOG: Resumen de características validadas en el proyecto PsiEngine para PsiModel. 154
Tabla 8-10. Proyecto PsiModel en PsiEnvironment. 154
Tabla 8-11. VOG: Resumen de características validadas en el proyecto PsiModel para PsiEngine y PsiModel. 156
Tabla 8-12. Proyectos de componentes de PsiDiagram en PsiEnvironment. 156
Tabla 8-13. VOG: Resumen de características validadas en los proyectos GLPsi, DPsi, TPsi, WPPsi y DFPsi para PsiEngine y PsiModel. 158

Tabla 8-14. Proyectos AngularSMAPPS. 159
Tabla 8-15. VOG: Resumen de características validadas en los proyectos de BPMEPsi, SMAPPS, AngularSMAPPS para PsiEngine y PsiModel. 162
Tabla 8-16. VOG: Número de validaciones de las características validadas para PsiEngine y PsiModel. 162
Tabla 8-17. Lista de proyectos PsiDiagram para la creación de diagramas, modelos gráficos y DVSLs... 163
Tabla 8-18. Proyectos diagramas de clases y diagramas PsiLSD de tipo PsiDiagram en PsiEnvironment.164
Tabla 8-19. VOG: Resumen de características/funcionalidades validadas con PsiEditDiagram. 168
Tabla 8-20. Proyectos ADG de tipo PsiDiagram en PsiEnvironment. 168
Tabla 8-21. VOG: Resumen de características validadas en los proyectos ADG. 171
Tabla 8-22. Proyectos SID de tipo PsiDiagram en PsiEnvironment. 171
Tabla 8-23. VOG: Resumen de características validadas en el Proyecto SID. 174
Tabla 8-24. Proyecto BPMEBase y proyectos adjuntos BPME. 174
Tabla 8-25. VOG: Resumen de características validadas en Proyecto BPME. 176
Tabla 8-26. VOG: Número de validaciones de las características para PsiDiagram. 177
Tabla 8-27. VOG: Número de validaciones de las características para PsiEngine, PsiModel y PsiModel. 178

Tabla C-1. Campo VALIDATOR de las estructura de datos para la Gramática GLPsi. 223
Tabla C-2. Campo VALIDATOR de la estructura de datos para la Gramática DPsi. 224
Tabla C-3. Campo VALIDATOR de las estructura de datos de la Gramática TPsi (Herramientas Visuales). 224
Tabla C-4. Campo VALIDATOR de las estructura de datos de la Gramática TPsi (Utilidades de Programación). 225
Tabla C-5. Campo VALIDATOR de las estructura de datos de la Gramática BPMEPsi. 226

Glosario de Acrónimos

- A -

ADG. Authomatic Diagram Generator

Anisha. Animación Anisha, lenguajes ShapesPsi y AnimePsi

- B -

BPM. Business Process Management

BPME. Business Process Management Engine

BPMEEditor. DSVL en el cliente web que genera programas BPMEPsi

BPMN. Business Process Management Notation

BPMEPsi. Business Process Management Engine Psi Language, Lenguaje BPM para aplicaciones Móviles

BPMVisor. Visualizador de configuración de Tipos de Proyectos BPME

- C -

CC[X]. Complejidad ciclomática (N=Número; M=Promedio; D=Densidad)

CSWE. Client-Side Web Environment

- D -

DOM. Document Object Model

DPsi. Diagramming Psi Language, Diagramación Psi

DSL. Domain Specific Languages

DSVL. Domain Specific Visual Language

- F -

FeedPsi. Aplicación Web FeedPsi, Lenguaje FeedPsi

- G -

GLPsi. Graphic Library Psi Language, Librería Gráfica Psi

GL[X]. Número de elementos gráficos (S=Figura Psi; L=Línea Psi; C=Contenedor Psi; A=Lista Psi)

GPVL. General Purpose Visual Languages

Grapher. Diagramación Grapher, lenguajes GraphsPsi y PaintPsi

- H -

H[X]. Métricas de complejidad de Halstead (N=Duración del programa; n=Tamaño del vocabulario; V=Volumen del programa; D=Nivel de dificultad; E=Esfuerzo de implementación; T=Tiempo de implementación; B=Estimación de errores)

HG. Número de generadores de diagramas Psi

HV[X]. Número de herramientas por visuales (D=Caja de diálogo; B=Barra de herramientas; T=Ventana emergente; F=Marco de contenido; R=Soltar; G=Arrastrar/Soltar; M=Menú; P=Menú flotante)

- I -

IDE. Integrated Development Environment

- J -

JSON. JavaScript Object Notation

- M -

MDA. Model-Driven Architecture

MDE. Model Driven Engineering

MI. Índice de mantenibilidad

MiniPsi. Lenguaje de Programación Ligero MiniPsi

MPsi. Modelo de Especificación Psi Language

MIPsi. Modelo de Implementación Psi Language

- N -

NC[X]. Número de elementos de una clase (P=Número de propiedades; M=Número de métodos; Z=Tamaño de la clase)

NHU. Número total de herramientas y utilidades Psi

NOP. Número de promedio de parámetros

- P -

PE[X]. Número de elementos de programación (V=Variables; F=Funciones; E=Tipos enumerados; C=Clases; O=Objetos; W=Widgets; S=Script)

PEC[X]. Número de elementos de una clase (P=Número de propiedades; M=Número de métodos; Z=Tamaño de la clase)

Psi. Programmable Solutions Interpreter

PsiCA. Psi Code Attribute

PsiCode. Código Fuente Psi

PsiData. Area común de intercambio de información entre programas Psi

PsiDiagram. Programmable Solutions Interpreter Diagram, Diagrama Programable Psi

PsiElement. Clase abstracta base para cualquier elemento de un lenguaje Psi

PsiEnvironment. Psi Development Environment, Entorno de Desarrollo Ligero Psi

PsiError. Error en programa Psi

PsiEngine. Programmable Solutions Interpreter Engine

PsiGVA. Psi Grammar Validator Attributes

PsiLI. Psi Link Information

PsiLSD. Psi Language Structure Diagram

PsiModel. Programmable Solutions Interpreter Model, Modelo de Programación Psi

PsiObject. Código Objeto Psi

PsiOut. Salidas de mensajes de programas Psi

PsiXML. PsiXML Interpreter

PsiWarning. Advertencia en programa Psi

PU[X]. Número de utilidades de programación (V=Variable; F=Función; M=Modificador gráfico; E=Skeleton Psi; C=Clases; L=Cargar de documento; K=Comando Psi)

- S -

SID. SCOOP Interface Diagram

SMAPPS. SCOOP Mobile Applications Service

SMS. Sistematic Mapping Study

SVG. Scalable Vector Graphics

- T -

TPsi. Tools Psi Language, Herramientas Psi (Herramientas Visuales, Utilidades de Programación, Generador de Diagramas, Recomendador de Diagramas)

TXml. Plantilla XML para PsiModel

- V -

VL. Visual Languages

VOG. Validación del Objetivo General

- X -

XMI. Unified Modeling Language

XML. Extensible Markup Language

XML-DSL. Domain Specific Languages con gramática XML

- W -

Widget. Aplicación pequeña que se visualiza a través de una ventana

Widget Web. Aplicación pequeña que se visualiza en un marco de una página

WPPsi. Web Page Psi Language, Plantillas de Páginas Web Psi

- [-

[X]LOC. Número de líneas (S=Código; L=Código Lógico; C=Comentarios; M=Promedio)

Capítulo 1. INTRODUCCIÓN

Este capítulo presenta una visión general de la tesis doctoral titulada "**Modelo programable para la serialización y evaluación de modelos heterogéneos en clientes web**". Se inicia con una motivación general y la descripción del problema, donde se resume los campos de investigación que originó esta tesis tales como aplicaciones web, los lenguajes de dominio específico, los intérpretes y los lenguajes XML en un cliente web. Como parte de este capítulo se define el alcance del trabajo estableciendo las hipótesis y los objetivos de investigación. Además, se presentan las principales contribuciones y se listan las publicaciones relacionadas a partir de la investigación realizada.

1.1 MOTIVACIÓN

Los cambios tecnológicos de la última década en el desarrollo de aplicaciones web han revolucionado la forma de concebir soluciones para los sistemas de información. En la actualidad, el diseño y la implementación de este tipo de soluciones siguen siendo temas vigentes de investigación. Las aplicaciones web se basan en la creación de páginas web dinámicas, en las cuales el lenguaje HTML maneja la distribución de contenido; el lenguaje CSS, la definición del estilo y la presentación; los lenguajes de scripting, DOM y AJAX, la funcionalidad y el dinamismo; y se pueden incluir imágenes, videos y/o audio usando tecnologías como PNG, JPEG y AVI (W3C, 2014; McDaniel, 2011; Crane, et al., 2005). Además, los lenguajes basados en XML como XSL, SVG, SMIL y MathML, también enriquecen el modelo de presentación de las páginas web (Fawcett, et al., 2012; Williams, 2009). Por otro lado, la Web 2.0 (Marín de la Iglesia, 2010; Anderson, 2012) aporta gran variedad de tecnologías, servicios y herramientas para la construcción de páginas web funcionales, agradables y usables, con la posibilidad de desplegarse en diferentes tipos de dispositivos.

Por otra parte, los Lenguajes de Dominio Específico (DSL, Domain Specific Languages) son lenguajes que ofrecen abstracciones de alto nivel para facilitar el diseño, la definición y/o la implementación de sistemas de información que den solución a problemas de un dominio específico (Fowler & White, 2010; Voelter, et al., 2013; Kleppe, 2008). Si un DSL se define con gramática XML (XML-DSL), permite que este sea fácil de usar, extensible y combinable, y pueda ser usado en diferentes plataformas y sistemas de información. Cuando un DSL se define mediante elementos gráficos se denomina Lenguaje Visual de Dominio Específico (DSVL, Domain Specific Visual Language). Los DSVLs permiten que expertos con conocimiento del dominio pero sin habilidades de programación puedan aportar al desarrollo de nuevos sistemas de información (Kelly & Tolvanen, 2008). Hasta el presente, la mayor parte de las propuestas de DSLs, XML-DSLs y DSVLs están dirigidas a solucionar problemas en

aplicaciones de escritorio, en entornos de desarrollo integrado (IDE, Integrated Development Environment) y en servidores web; y la creación de los correspondientes DSLs, XML-DSLs y DSVLs es implementada usualmente en los IDEs (Fowler & White, 2010; Ghosh, 2010; Granback, 2009; Bettini, 2013; Rahien, 2010).

En clientes web, las propuestas de creación de XML-DSLs y DSVLs son bastante escasas. Por un lado, los XML-DSLs suelen usar transformaciones XSLT para obtener resultados de presentación o transformación de información XML. Por otro lado, la mayoría de aplicaciones web y *frameworks* están dirigidas a la elaboración de diagramas y modelos gráficos[1], y la visualización de información heterogénea[2] en línea. Es decir, no hay entornos de modelado visual o herramientas visuales para la creación y evaluación de DSVLs basadas en las tecnologías de un cliente web. Por tanto, esta es un área en la que, conociendo las limitaciones de un cliente web, hay margen para la investigación y la innovación para el desarrollo de un *modelo programable para la definición, serialización y evaluación de DSVLs*.

Por otra parte, la Ingeniería Dirigida por Modelos (MDE, Model Driven Engineering), es una metodología de desarrollo de software centrada en la creación de modelos sobre dominios específicos para simplificar el proceso de creación de sistemas de información (Schmidt, 2006; Nolan, et al., 2008). Este paradigma combina los conceptos de DSLs, XML-DSLs y DSVLs con motores de transformación de modelos y generación de código.

En este contexto, B2T Concept S.L. (b2tconcept.com) es una compañía española especializada en la gestión de procesos de negocio y gestión del conocimiento para todo tipo de empresas y sectores. En B2T se desarrolló la Plataforma B2T SCOOP. Esta plataforma es una aplicación web encaminada a la creación de soluciones de negocio basadas en el modelado de Gestión de Procesos de Negocio (BPM, Business Process Management) y en el metamodelado recursivo de negocio, denominado Enterprise Recursive Meta-Modeling (ERMM) (Díez, et al., 2013). Además, la Plataforma SCOOP hace uso de la metodología de MDE definiendo *modelos heterogéneos* (modelos basados en DSLs, XML-DSLs, DSVLs, diagramas y modelos gráficos) tales como: diagramas de clases UML, modelo de estado, modelo entidad relación, estructuras organizativas, administración de proyectos, BPMN v2.0 (OMG, 2011), entre otros.

Para dar soporte gráfico a los diferentes *modelos heterogéneos* usados en la Plataforma SCOOP, es necesario un entorno de modelado visual programable que posibilite la creación de XML-DSLs, DSVLs y la generación de diagramas con capacidad de asociación de información heterogénea XML/JSON. La Plataforma SCOOP es el entorno perfecto para la explotación y validación de los resultados obtenidos en esta tesis.

1.2 DESCRIPCIÓN DEL PROBLEMA

Partiendo de la motivación anterior, el problema a resolver en esta tesis es *crear* y *evaluar* XML-DSLs y DSVLs y *generar* diagramas con capacidad de asociar información heterogénea XML y/o JSON para el desarrollo de modelos en clientes web, es decir, la creación de un modelo programable que permita la serialización y evaluación de modelos heterogéneos en clientes web. En este contexto, por un lado, *crear* hace referencia a definir la gramática XML y asociar a cada etiqueta una funcionalidad en JavaScript, y *evaluar* permite tomar un programa escrito con esta gramática XML y ejecutar su funcionalidad JavaScript. Por otro lado, *generar* diagramas implica la construcción de XML-DSLs con capacidad de

[1] Por ejemplo Gliffy (gliffy.com), LucidChar (lucidchart.com), yFiles (yworks.com), GoJS (nwoods.com) y Cacoo (cacoo.com).

[2] Podemos mencionar a Graphviz (graphviz.org), Highcharts (highcharts.com), D3.js (d3js.org), Google Charts (developers.google.com/chart), Protovis (mbostock.github.io/protovis/), entre otros. Estas herramientas son vistosas y fáciles de usar, y se basan en información JSON con una estructura predefinida.

asociar información XML/JSON y algoritmos de diagramación sobre el cliente web. Para ello se requiere crear un *entorno de modelado visual programable*, consiste en un intérprete especializado con capacidad de crear librerías gráficas, pintar elementos gráficos basados en estas librerías, y con capacidad de asociar funcionalidades visuales (menús, cajas de diálogo, ventanas emergentes, etc.) y asociar información heterogénea XML y/o JSON.

Esta sección reflexiona y profundiza sobre los DSLs, XML-DSLs, DSVLs y el Proyecto SCOOP para resolver el problema planteado y proponer los objetivos de investigación de esta tesis y su respectiva validación

1.2.1 Lenguajes de Dominio Específico

Los Lenguajes de Dominio Específico (DSL), como se ha mencionado antes, son lenguajes de programación especializados que ofrecen abstracciones y técnicas de alto nivel para facilitar la comprensión y el funcionamiento de un dominio específico o parte de él. Su objetivo es facilitar el diseño, la definición y la implementación de sistemas de información para dar solución a problemas de dominios específicos, permitiendo a los expertos del área realizar las tareas de manera más eficiente, y produciendo sistemas de mayor calidad y fiabilidad (Fowler & White, 2010; Voelter, et al., 2013; Kleppe, 2008). Un DSL puede ser descrito haciendo uso del metamodelado, mediante el cual se definen relaciones entre los conceptos de un dominio y se especifican con precisión la semántica y las restricciones asociadas con estos conceptos. Es fácil pensar que la construcción de un DSL es una labor costosa en términos de programación, pero es más difícil modelar la gramática del DSL y lograr una cohesión armónica entre la gramática, su funcionalidad y la solución que aporta a un problema específico (Fowler & White, 2010).

La creación de un DSL requiere definir la gramática del lenguaje y su funcionalidad asociada. Además, requiere crear un intérprete o compilador para que tome un programa escrito en el lenguaje fuente DSL, unos datos y unos recursos, los analice y los evalúe, para obtener un resultado o solución. Un compilador, a diferencia del intérprete, transforma el código fuente a código objeto (proceso de compilación), el cual se ejecuta tomando ciertos datos de entrada y generando los resultados (Ruiz, 2010; Alfonseca, et al., 2006). Los intérpretes y compiladores suelen crearse aprovechando la existencia de herramientas como Lex & Yacc (Brown, et al., 1992) o Flex & Bison (Levine, 2009), y su uso es extensivo a nivel de aplicaciones de escritorio y servidores. Los IDEs como Eclipse, Visual Studio y ASF+SDF Meta-Environment también ofrecen herramientas y lenguajes para la implementación de DSLs.

Para usar DSLs en los clientes web, hemos recurrido a la creación de un intérprete especializado usando las tecnologías disponibles en este contexto. Pueden encontrarse propuestas como Jison que genera analizadores sintácticos ascendentes en JavaScript basados en las principales características de Bison (Carter, 2009), es decir, DSLs textuales implementados en JavaScript. El intérprete especializado que proponemos en esta tesis va dirigido hacia la evaluación de XML-DSLs con capacidad de asociar información heterogénea XML y/o JSON.

1.2.2 Lenguajes de Dominio Específico basados en gramáticas XML

El Lenguaje de Marcado Extensible (XML, Extensible Markup Language) supone un estándar para almacenar información autodocumentada y el intercambio de información estructurada entre sistemas de información (W3C: XML, 2008; W3C: XML Technology, 2010).

A nivel de servidor web, los lenguajes como Web Forms de ASP.NET (Microsoft, 2014), Java Server Pages (Oracle, 2003), Java Server Faces (Oracle, 2014) y WebDSL (Visser, 2008; Groenewegen & Visser, 2009) tienen como objetivo de crear aplicaciones web a nivel de servidor, integrando modelos de datos, modelos de interfaz de usuario y control de acceso.

A nivel de cliente web, como metalenguaje XML permite definir diferentes lenguajes como XSL, SVG, SMIL y MathML, que enriquecen el modelo de presentación de una página web y son interpretados de forma nativa por la mayoría de los navegadores actuales (Fawcett, et al., 2012; Williams, 2009). Cada lenguaje XML tiene un objetivo específico, y cada elemento del lenguaje tiene una funcionalidad específica asociada, que contribuye a este objetivo. En general, el modelo de objetos de documento DOM accede y modifica la estructura y el contenido HTML o XML a través de lenguajes con especificación ECMAScript (por ejemplo, JavaScript), y en particular, cada lenguaje XML que soporte un cliente web, tiene implementado un DOM.

En resumen, para implementar nuevos XML-DSLs en un cliente web, se requiere que sean procesados como documentos DOM y que su funcionalidad sea programada con JavaScript. Además, si los nuevos XML-DSLs se combinan con componentes, widgets, frameworks y lenguajes XML, y se aplican políticas de seguridad y buenas prácticas de programación (Yue & Wang, 2013; Kern, 2014), se puede crear soluciones rápidas, seguras, robustas y flexibles.

En este contexto, el planteamiento general de la tesis se orienta hacia la implementación de un *intérprete especializado* de XML-DSLs en JavaScript que evalúe programas en cualquier navegador y dispositivo. Este intérprete debe ser liviano, genérico y debe permitir definir múltiples gramáticas e interpretar múltiples programas, con capacidad de coexistencia e intercambio de información, y además, que pueda asociar y operar contenidos de múltiples fuentes heterogéneas XML/JSON de forma nativa. En términos generales, un XML-DSL creado para este intérprete se fundamenta en definir un conjunto de etiquetas programadas, es decir, a cada etiqueta se le asocia una funcionalidad en lenguaje JavaScript. La evaluación de un programa escrito en este nuevo XML-DSL, es la ejecución de cada funcionalidad asociada a cada etiqueta. Con estas características, se pueden diseñar XML-DSLs más robustos, creativos y con infinidad de posibilidades para construir componentes, widgets y/o aplicaciones web.

1.2.3 LENGUAJES VISUALES DE DOMINIO ESPECÍFICO

Los Lenguajes Visuales de Dominio Específico (DSVL) son representaciones gráficas que permiten definir un conjunto de elementos gráficos que describen formalmente los conceptos de un dominio o parte del él (Kelly & Tolvanen, 2008). Existen diversos trabajos y herramientas para definir DSVLs, en (Guerra, 2007) se propone una herramienta de metamodelado que es usada para generar entornos de DSVLs en diversos dominios. De manera similar, (Bottini & Grau, 2004; Bottini, et al., 2010) especifican un conjunto de metamodelos para la definición y gestión de lenguajes visuales, mientras que (Heer & Agrawala, 2006) proponen un conjunto de patrones de diseño de software para la visualización de modelos de datos estructurados.

Como ejemplos de DSVLs, se pueden mencionar: Web Modeling Language (WebML), que define una notación visual para el diseño de aplicaciones web (Mishra & Mohanty, 2011), y Business Process Modeling Notation (BPMN) un lenguaje visual para el modelado de procesos de negocios basado en el concepto de flujo de trabajo (Pant & Juric, 2008; OMG, 2011).

En la actualidad, existen potentes herramientas de escritorio y entornos de desarrollo de DSVLs que facilitan tanto la definición del lenguaje visual como la lógica necesaria para la transformación a código fuente. Entre las primeras podemos destacar: GMF y Graphiti (Steinberg, et al., 2008), Visual Studio Visualization and Modeling VSSDK (Microsoft, 2015) y VLDesk (Costagliola, et al., 2004). Sin embargo, las herramientas y entornos de desarrollo de DSVLs para clientes web, no alcanzan un desarrollo tecnológico como las de escritorio, están dirigidas a la elaboración visual de modelos y diagramas en línea. Se pueden mencionar: Gliffy (gliffy.com), LucidChar (lucidchart.com), yFiles (yworks.com), GoJS (nwoods.com) y Cacoo (cacoo.com). Estas herramientas web se basan en manipular y administrar documentos SVG, canvas HTML o Adobe Flash, principalmente, y usan librerías o plantillas gráficas para su fácil edición. Además, carecen de utilidades o estrategias para la generación de modelos

gráficos o diagramas a partir de fuentes de información heterogénea como XML y/o JSON. Usualmente, este tipo de herramientas establecen estructuras JSON predefinidas para este tipo generación.

Para aportar nuevas posibilidades y riquezas al conjunto de los DSVLs existentes para clientes web, en esta tesis se propone la definición de un diagrama programable basado en la implementación de XML-DSLs y evaluados sobre nuestro *intérprete especializado*. El objetivo de este nuevo diagrama programable es el de definir librerías gráficas generadas a partir de SVG, serializar modelos gráficos y DSVLs con gramática XML y añadir funcionalidades y herramientas (cajas de diálogo, barras de herramientas, ventanas emergentes, menús, menús flotantes, soltar, arrastrar y soltar, etc.) para enriquecer su construcción en los clientes web. Los modelos gráficos y DSVLs creados a partir del diagrama programable, permiten su modificación en tiempo real y que su apariencia y contenido puedan depender de fuentes heterogéneas de información como XML o JSON. Adicionalmente, se incluyen funcionalidades para la generación automática de modelos gráficos y un modelo de recomendación para ésta generación.

1.2.4 Trabajos previos de creación de DSLs Textuales, XML-DSLs y DSVLs

En esta sección se resumen y analizan los aportes de tres trabajos previos que nos ayudaron y motivaron a plantear los objetivos de la presente tesis. Para cada trabajo se presenta un resumen y se describen que tipo de DSL (DSL Textual, XML-DSL o DSVL).

T1. Sistema para la Generación Automática de Contenido y Evaluación para e-Learning, basados en el Uso de Herramientas XML

En (Chavarriaga & Díez, 2008) se define un sistema para la generación automática de contenidos y evaluaciones para e-Learning, cuyo objetivo es crear una página web dinámica en el servidor con contenidos en notación matemática MathML y gráficos en SVG, JPEG, PNG o GIF, para el área de las Matemáticas. Para cumplir este objetivo se crearon los siguientes lenguajes específicos:

- *Lenguaje para el contexto de trabajo y ejecución*: es un XML-DSL para especificar un conjunto de expresiones y funciones matemáticas con gramática XML y lenguaje de Mathematica®. Además, especifica una interfaz de ejecución ligera conectada al motor Mathematica® para obtener resultados en MathML, HTML, SVG, JPEG, PNG y/o GIF, e incluye un lenguaje textual para implementación de funciones de Mathematica®.
- *Especificación de evaluaciones*: es un XML-DSL que define la estructura de una evaluación basada en preguntas de tipo: explicación, falso y verdadero, selección única, selección múltiple, ordenación, correspondencia entre listas, entre otras. Hace uso de XSLT, CSS y el lenguaje de trabajo y ejecución para generar una evaluación automática de matemáticas.

T2. Arquitectura de un sistema para la generación automática de contenidos para evaluación basados en el uso de especificaciones XML

Partiendo del trabajo (Chavarriaga & Díez, 2008) anterior, en el Trabajo de Fin de Máster (Chavarriaga, 2009), se formalizó una arquitectura para un sistema de generación automática de contenidos para las evaluaciones académicas. El objetivo del trabajo fue proveer en el servidor web de componentes adaptables (*pluggins*), basados en XML-DSLs para la administración y generación de evaluaciones, donde un usuario solicite una evaluación con ciertas características y el componente genere de forma automática la evaluación en lenguaje **IMS QTI** (IMS Question & Test Interoperability Specification (IMS QTI, 2008)). Para obtener ésta generación automática se creó:

- *Lenguaje MGA-QTI*: es un XML-DSL para la generación y creación del manifiesto de evaluación IMS QTI.
- *Lenguaje MGQ-QTI*: es un XML-DSL para la generación de preguntas basadas en asociar fuentes de información XML. Implementa los tipos de preguntas: selección simple, selección múltiple, ordenación y asociación.

T3. Nuevos Modelos de Presentación para la Visualización de Semántica en Interfaces de Usuario Modernas

En (Chavarriaga & Macías, 2009; Chavarriaga & Macías, 2008) se presenta una nueva forma de visualizar información del dominio proveniente de la Web Semántica a través de técnicas y elementos de programación basados en el uso de transformaciones XSLT y de la Web 2.0, aportando una arquitectura que permite una mayor interacción del usuario final con interfaces de usuario Web más expresivas y ricas en información. En este trabajo se crearon los siguientes lenguajes:

- *Página Maestra*: define un DSL textual para especificar una plantilla HTML dividida en secciones para ser reemplazada, ya sea, por fragmentos de página, menús, imágenes, videos, *applets*, etc.
- *Lenguaje de Modelado Web* (LMW): es un DSVL que define un lenguaje de navegación entre páginas web agrupados por conceptos y definición páginas maestras.

La Tabla 1-1 resume la cantidad de DSLs textuales, XML-DSLs y DSVLs de los trabajos previamente mencionados. En total, se construyeron un DSL textual, cuatro XML-DSLs y un DSVL para un IDE. Es de notar que los XML-DSLs fueron desarrollados para ser utilizados como solución, al lado del servidor web. De estos trabajos surgió la necesidad de establecer reglas para asociar información XML a ciertos elementos del lenguaje, y se estableció que la funcionalidad para realizar esta asociación es muy parecida. De aquí también nace la idea de crear un intérprete especializado para evaluar programas escritos en XML-DSLs, que soporte asociación de información y ejecución de código *inline*, pero que esta evaluación se realice directamente al lado del cliente web.

Tabla 1-1. Resumen de trabajos previos de creación DSLs textuales, XML-DSLs, DSVLs.

Trabajo	DSL Textual	XML-DSL	DSVL	Servidor	Cliente	IDE
T1	1	2	0	☑	☐	☐
T2	0	2	0	☑	☐	☐
T3	0	0	1	☐	☐	☑
Total	1	4	1			

1.2.5 MDE y Plataforma B2T SCOOP

Un *modelo* es una representación abstracta, conceptual o gráfica de procesos o problemas de un dominio específico, con el fin de analizarlos, describirlos, explicarlos y/o solucionarlos. Distintos *modelos* pueden definir distintos componentes y funcionalidades que conforman el dominio (Mellor, et al., 2003; Schmidt, 2006). En Ingeniería de Software, la Arquitectura Dirigida por Modelos (MDA, Model-Driven Architecture) es la metodología de software centrada en la definición de modelos para simplificar el proceso de creación de sistemas de información propuesto por el Object Management Group (OMG). El término “arquitectura" en MDA está relacionado con los distintos estándares y normas que sirven como base tecnológica para la especificación e implementación de los sistemas de información, entre los que se incluyen: Unified Modeling Language (UML), Meta-Object Facility (MOF), XML Metadata Interchange

(XMI), Enterprise Distributed Object Computing (EDOC), el Software Process Engineering Metamodel (SPEM) y el Common Warehouse Metamodel (CWM).

La Ingeniería Dirigida por Modelos (MDE, Model Driven Engineering) tiene los mismos principios que MDA relacionados con la ingeniería de modelos, pero es utilizado por la comunidad investigadora sin centrarse exclusivamente en los estándares de OMG. La MDE combina los conceptos de DSLs, XML-DSLs y/o DSVLs con motores de transformación de modelos y generación de código.

En B2T se desarrolló un motor de reglas de negocio ejecutable denominado E3, que se utiliza para la creación de soluciones de negocio basadas en el modelado de dominios de negocio y procesos. La Plataforma B2T SCOOP utiliza E3 para la creación de modelos formales encaminados a la representación del conocimiento de empresas. Su objetivo es ayudar a crear sistemas de información para estas empresas a partir de dichos modelos formales. La Plataforma B2T SCOOP hace uso de la metodología de MDE y tiene la posibilidad de definir *modelos heterogéneos* a diferentes niveles: en la generación automática de modelos en la parte de diseño (por ejemplo, modelo de clases, de estado, de entidad-relación, de llamadas a funciones, diagramas de GANT, de PERT etc.), en la visualización de dominios específicos del negocio (por ejemplo, estructuras organizativas, administración de proyectos, seguridad, etc.), en la creación de aplicaciones móviles híbridas, y por último, en el soporte de BPMN v2.0 (OMG, 2011).

Mi aportación en la creación de diferentes modelos y diagramas gráficos en la Plataforma SCOOP, resultó idóneo para la explotación y validación de los resultados obtenidos en esta tesis.

1.3 Objetivos de investigación

Para contextualizar el marco de investigación desarrollado en esta tesis es necesario establecer un entorno general de trabajo, al que denominaremos Entorno Cliente Web (CSWE, Client-Side Web Environment). Este entorno consiste en el conjunto de *tecnologías*, *lenguajes*, *metodologías de programación* y *frameworks* disponibles en el navegador o cliente web para la implementación de componentes, widgets o aplicaciones web.

Como se ha indicado anteriormente, esta tesis se centra en la definición, desarrollo y evaluación de DSL con gramática XML (XML-DSL) con la capacidad de asociar información heterogénea (XML/JSON) que ayude a dar solución a un problema específico o parte de él en clientes web. A continuación se presentan las hipótesis de investigación que se van a explorar en el desarrollo de esta tesis:

H1. Es posible definir e implementar un intérprete que evalúe múltiples XML-DSLs en el CSWE.

H2. Es posible asociar información heterogénea XML y/o JSON de forma nativa en los XML-DSLs en el CSWE.

H3. Es posible crear un modelo de programación basado en XML-DSLs en el CSWE, para facilitar la definición de nuevos XML-DSLs.

H4. El diagrama programable permite crear modelos gráficos, generar diagramas y crear DSVLs en el CSWE.

H5. El uso de múltiples XML-DSLs junto con la evaluación de modelos heterogéneos dan solución a problemas de dominio específico en el CSWE.

H6. Los mecanismos de recomendación para la generación de diagramas facilitan al usuario una mejor visualización para los modelos gráficos y diagramas.

Con la motivación, la descripción del problema y las hipótesis de investigación se plantea el objetivo general de esta tesis:

OBJETIVO GENERAL

La especificación, creación y evaluación de XML-DSLs con capacidad de asociar información heterogénea (XML/JSON) en un cliente web, para la construcción de componentes, widgets, XML-DSLs, DSVLs, modelos gráficos y diagramas que ayuden a dar soluciones a problemas específicos.

Para conseguir este objetivo general se requiere crear un motor de evaluación en el CSWE de programas escritos en XML-DSL. Junto con el motor, se implementarán lenguajes para proveer un modelo de programación que permita la creación de XML-DSLs mencionados. Para demostrar las posibilidades de desarrollo y la potencialidad tanto del motor como de los lenguajes desarrollados en el CSWE, se establecieron tres lenguajes para la elaboración de modelos visuales programables con el objetivo de crear modelos visuales, diagramas y DSVLs. Adicionalmente, se propone una generación automática de diagramas y un recomendador de diagramas en clientes web.

Para asegurar el logro del objetivo general, se han planteado cuatro objetivos específicos, los cuales suponen, por un lado, una descomposición exhaustiva del objetivo general y, por otro, permitieron definir claramente las actividades a realizar en cada una de las etapas de la investigación.

O1. Definir y desarrollar un motor de evaluación de programas escritos en XML-DSL con capacidad de asociar información heterogénea (XML/JSON) en un CSWE.

El motor de evaluación de programas escritos en XML-DSL que se propone se implementará en lenguaje JavaScript. Este motor debe ser liviano, genérico (que evalúe en cualquier navegador y dispositivo móvil) y permitan definir múltiples XML-DSLs e interpretar múltiples programas, con capacidad de coexistencia e intercambio de información heterogénea (XML/JSON) en un CSWE. Para este objetivo se realizó una revisión y análisis de las herramientas y entornos de desarrollo para la creación y evaluación de DSL y en particular para XML-DSL (Capítulo 2). Además, se revisó el tipo de intérpretes o frameworks que se manejan en un CSWE, y se hizo un análisis, diseño e implementación del motor de evaluación (Capítulo 3) y se crearon un conjunto de casos de estudio para su validación (Capítulo 8).

O2. Definir y desarrollar un modelo de programación para la creación de XML-DSLs en un CSWE.

Este segundo objetivo específico se orienta al diseño de un modelo de programación con generación de código JavaScript que facilite la creación de componentes, widgets y XML-DSLs sobre el motor de evaluación del O1 en el CSWE. Este modelo de programación busca eficiencia en la construcción de XML-DSLs, que se traduce en ahorro de tiempo y aumento en calidad, estandarización y fiabilidad del sistema de información. Para este objetivo se realizará una revisión de herramientas y entornos de desarrollo para la creación de un nuevo modelo de programación en CSWE (Capítulo 2) y se abordará una nueva propuesta en el Capítulo 4. Su validación se hará en los Capítulos 6 y 7.

O3. Definir y desarrollar una propuesta que facilite el diseño e implementación de diagramas programables para la creación de DSVLs y la generación automática de diagramas en un CSWE.

En el tercer objetivo específico se realizará una propuesta basada en librerías gráficas SVG y en el uso de XML-DSLs sobre el motor de evaluación del O1, para la creación de diagramas programables, donde su apariencia y contenido puedan depender de múltiples fuentes heterogéneas de información (XML/JSON) en un CSWE, es decir, diagramas que se adapten a la información. Además, estos diagramas

deberán tener la capacidad de generar elementos gráficos y asociar funcionalidades y herramientas como cajas de diálogo, barras de herramientas, ventanas emergentes, menús, soltar, arrastrar y soltar, etc. Con estas características, el diagrama programable tendrá los elementos necesarios para construir DSVLs y generar automáticamente diagramas en un CSWE. Para este objetivo se requerirá revisar y analizar las herramientas y entornos de desarrollo para la creación de diagramas y DSVLs en un CSWE (Capítulo 2) y se abordará una nueva propuesta en el Capítulo 5. Su validación se realizará en el Capítulo 8.

O4. Crear mecanismos de recomendación gráfica a partir de la generación automática de diagramas del O3.

Para desarrollar este objetivo específico, se identificarán qué tipos de mecanismos de recomendación ayudan a los desarrolladores de diagramas en un CSWE para mejorar su visualización. Este objetivo se abordará en el Capítulo 5, como parte de la funcionalidad de la propuesta de diagramas programables.

1.4 RESULTADOS

Las contribuciones de esta tesis se evidencian de dos formas: como parte de la transferencia tecnológica aportada a la Plataforma SCOOP; y a través de un conjunto de publicaciones en revistas y congresos.

1.4.1 TRANSFERENCIA TECNOLÓGICA

Las propuestas del modelo de programación (O2) y el diagrama programable (O3) en un CSWE aporta a la Plataforma B2T SCOOP cuatro tipos de escenarios: *Proyecto ADG* (Authomatic Diagram Generator), *Proyecto SID* (SCOOP Interface Diagram), *Proyecto BPME* (Business Process Management Engine) y *Proyecto SMAPPS* (SCOOP Mobile Applications Service).

El ***Proyecto ADG*** surge para resolver los problemas de visualización de diagramas y de información a partir de fuentes heterogéneas en el sistema SCOOP de B2T. Es un conjunto de programas escritos siguiendo la propuesta de diagrama programable de la presente Tesis. Se emplea para la generación de modelos gráficos de forma automática basados en información heterogénea del sistema SCOOP. Los tipos de diagramas implementados son: diagrama de GANT, diagrama de PERT, BPM para SCOOP, conceptos y relaciones, diagrama de estados, modelo entidad-relación, representación gráfica de cohesión de módulos, representación gráfica de objetos y otras representaciones internas. Cada tipo de diagrama tiene diferentes representaciones y en consecuencia, se pueden aplicar mecanismos de recomendación para obtener la visualización más apropiada del diagrama (O4).

El ***Proyecto SID*** es una evolución del *Proyecto ADG* que cubre la sincronización entre la edición de diagramas y la información preexistente en la plataforma SCOOP. Al igual que ADG, es un conjunto de programas escritos siguiendo la propuesta de diagrama programable. Actualmente están implementados los siguientes diagramas:

- Para el Proyecto *Modelo Técnico*: modelo de conceptos-relaciones y modelo de estados.
- Para el Proyecto *Sketch*: modelo de conceptos-relaciones, modelo de estados y modelos de procesos.
- Para el Proyecto *BPMs y Eventos*: modelo de BPM.

El ***Proyecto BPME*** resuelve la creación y edición de procesos de negocios basado en la especificación BPMN 2.0 (OMG, 2011) y puede ser usado por SCOOP, cualquier aplicación web o widget. Al igual que los anteriores es un conjunto de programas escritos siguiendo la propuesta de diagrama

programable. Este proyecto cumple con la especificación de elementos gráficos dispuestos en el estándar BPMN 2.0 y sirve como base para proyectos que se construyen bajo la filosofía de un proceso de negocios. Los proyectos implementados con esta filosofía son:

- Diagramas SMAPPS.
- DSVL para la creación de servicios de aplicaciones móviles basados en un subconjunto de BMPN 2.0.
- Generación automática de diagramas BPM para BPMs y Eventos.
- Visor de Proyectos BPMEs.

Por último, el ***Proyecto SMAPPS*** permite generar el código de una aplicación móvil a partir de la definición de servicios SMAPPS de la plataforma SCOOP (DSVLs y fuentes heterogéneas de datos). Al igual que los anteriores es un conjunto de programas escritos siguiendo la propuesta de diagrama programable para la creación, edición y ejecución de aplicaciones móviles creadas con el *Proyecto BPME* de SCOOP. Este proyecto se divide en dos partes: definición de *servicios SMAPPS* en SCOOP y *aplicaciones móviles híbridas* basada en *servicios SMAPPS*. Un *servicio SMAPPS* se especifica mediante un proceso de negocios usando una versión reducida y especializada de BPMN 2.0, para modelar la navegación entre páginas, la renderización de páginas, la ejecución de scripts, la conexión con servicios web y el intercambio de información entre *servicios SMAPPS*. *SMAPPS* tiene la opción de generar todos los *servicios* comprimidos en un fichero. Además, en B2T Concept se desarrolló una arquitectura en iOS y Android, basada en la filosofía de una aplicación híbrida usando el framework cordova (cordova.apache.org), con la capacidad de interpretar y ejecutar los *servicios SMAPPS*, es decir, ejecutar las *aplicaciones móviles hibridas*. El corazón para ejecutar un *servicio SMAPPS* es un XML-DSL sobre nuestro motor de evaluaciones (O1) y un renderizador de páginas basado en una especificación JSON.

Todos estos proyectos se detallarán en el Capítulo 7 y se validarán en el Capítulo 8.

1.4.2 PUBLICACIONES

Las contribuciones de esta tesis se reflejan en un conjunto de publicaciones que clasificaremos en dos grupos diferenciados: revistas y congresos.

1.4.2.1 Revistas

Listado de contribuciones en revistas:

(R1) **CHAVARRIAGA, E.**, JURADO, F., DÍEZ, F. (2015). *An Approach to Build XML-based Domain Specific Languages Solutions for Client-Side Web Applications*. Computer Languages, Systems & Structures, Elsevier (Aceptado), pp. 1-23.

En este trabajo se formaliza el motor de evaluación de programas escritos con XML-DSL con capacidad de asociar información heterogénea (XML/JSON) en un cliente web. Se describe una metodología para la creación de XML-DSLs en un CSWE. Por último se detallan dos casos de estudio: Anisha y FeedPsi. Este trabajo es la consolidación del O1.

(R2) **CHAVARRIAGA, E.** Y MACÍAS, J. (2009). *A model-driven approach to building modern Semantic Web-Based User Interfaces*. Advances in Engineering Software, *12*(40), pp. 1329-1334.

En este trabajo se presentó una nueva forma de visualizar información del dominio proveniente de la Web Semántica a través de técnicas y elementos de programación basados en el uso de transformaciones XSLT y de la Web 2.0, aportando una arquitectura que permita una mayor interacción del usuario final con interfaces de usuario web más expresivas y ricas en información. Este trabajo es una extensión de (C4) y sirvió para explorar las características que ofrecen la Web 2.0 y el uso de

transformación de información con XSLT y la creación de un DSL y un DSVL con herramientas de escritorio. Además, el trabajo nos aportó experiencia para desarrollar el O2 y O3.

(R3) BELLOGÍN, A., CANTADOR, I., DÍEZ, F., CASTELLS, P. Y **CHAVARRIAGA, E.** (2013). *An Empirical Comparison of Social, Collaborative Filtering, and Hybrid Recommenders*. ACM Transactions on Intelligent Systems and Technology, Special Issue on Context-Aware Movie Recommendations 4 (1), 14, pp. 1-29, Febrero 2013.

El objetivo del trabajo es proponer una métrica de cobertura que declara y compensa el carácter incompleto de las evaluaciones de desempeño basadas sólo en la precisión. Se utilizó esta medida, junto con métricas de precisión para realizar una comparación empírica de varios filtrados sociales, colaborativos y recomendadores híbridos. En particular, mi aporte fue la creación de algoritmos para el filtrado colaborativo basados en caminos aleatorios en un contexto social. Este trabajo nos aportó, adicionalmente, conocimientos y criterios para la creación de recomendaciones en diferentes tipos de escenarios, aspecto éste relacionado con el O4.

1.4.2.2 Congresos

A continuación se listan los congresos y conferencias:

(C1) **CHAVARRIAGA, E.**, DÍEZ, F. Y DÍEZ, A. (2012). *Intérprete PsiXML para Mini-Lenguajes XML en aplicaciones Web*. Actas de las "XII Jornadas sobre Programación y Lenguajes (PROLE)", Sistedes 2012, p. 213-227. Almería, España, 17 al 19 de Septiembre de 2012.

En este trabajo se hizo una primera formalización del *Intérprete PsiXML* (O1). Se introdujo el concepto de XML-DSL en CSWE como mini-lenguajes XML. Como caso de estudio se construyó una aplicación web para el manejo de fuentes RSS de información, partiendo de un mini-lenguaje denominado FeedPsi.

(C2) **CHAVARRIAGA, E.** Y DÍEZ. F. (2008) *Sistema para la Generación Automática de Contenido y Evaluación para e-Learning, basados en el Uso de Herramientas XML*. En C. Vaz de Carvallho, M. Llamas, and R. Silveira (Eds.), TICAI 2008: TICs para el Aprendizaje de la Ingeniería, Capítulo 8, pp. 53-60 ©IEEE, Sociedad de Educación: Capítulos Español y Portugués. ISBN 978-972-8688-63-9

Este trabajo define un sistema para la generación automática de contenido y evaluación para e-Learning especializado en el área de matemáticas, basados en especificar un XML-DSL para la definición de evaluaciones de matemáticas, y un XML-DSL para especificar una interfaz de ejecución con el motor de Mathematica®, obteniendo como resultado notación matemática en MathML, gráficos en SVG e imágenes JPEG, PNG, GIF. Este trabajo nos aportó experiencia en la creación y evaluación de XML-DSLs en un servidor web, relacionado con los objetivos específicos O1, O2 y O3.

(C3) DIEZ, A., NGUYEN, N., **CHAVARRIAGA, E.**, DIEZ, P. Y DIEZ, F. (2013). *Organizations as Models, Models as Systems: the ERMM Theory of Large Scale Enterprise Executable Models*. MODELSWARD 2013. 1st International conference on Model-Driven Engineering and Software Development, February 19-21, 2013, Barcelona, España.

En este trabajo se combinó el metamodelado de dominio específico, con lenguajes de dominio específico, repositorio e intérpretes de modelos para crear una nueva teoría sobre metamodelado recursivo de negocio, denominado Enterprise Recursive Meta-Modeling (ERMM). Este trabajo nos aportó criterios y nuevas perspectivas para la creación de modelos a partir de otros modelos.

(C4) **CHAVARRIAGA, E.** Y MACÍAS, J. (2008). *Nuevos Modelos de Presentación para la Visualización de Semántica en Interfaces de Usuario Modernas*. IX Congreso de Interacción Persona-Ordenador

(Interacción 2008), pp. 1-10, Albacete, España.

En este trabajo se presentó una nueva forma de visualizar información del dominio proveniente de la Web Semántica a través de técnicas y elementos de programación basados en el uso de transformaciones XSLT y de la Web 2.0. Se definió un DSL textual para especificar una plantilla HTML dividida en secciones para ser reemplazada, ya sea, por fragmentos de página, menús, imágenes, videos, applets, etc., y un lenguaje de modelado web (LMW), es un DSVL que define la navegación entre páginas web y es agrupado por conceptos y definición de páginas maestras.

(C5) DÍEZ, F., **CHAVARRIAGA, E.**, CAMPOS, P. Y BELLOGÍN, A. (2010). *Movie Recommendations based in explicit and implicit features extracted from the Filmtipset dataset*. Proceedings of the Workshop Challenge on Context-aware Movie Recommendation (CAMRa 2010), desarrollado conjuntamente con 4th ACM Conference on Recommender Systems 2010 (RecSys 2010), pp. 45-52, Barcelona, España.

El objetivo del trabajo es utilizar diferentes algoritmos de filtrado colaborativo basados en caminos aleatorios para explotar información de contexto social en la forma de relaciones de amistad en un conjunto de datos de ratings de películas. Al igual que (R3), este trabajo nos aportó conocimientos y criterios para la creación de recomendaciones en diferentes tipos de escenarios.

(C6) CAMPOS, P., BELLOGÍN, A, DÍEZ, F. Y **CHAVARRIAGA, E.** *Simple Time-Biased KNN-based recommendations*. Workshop Challenge on Context-aware Movie Recommendation (CAMRa 2010), desarrollado conjuntamente con 4th ACM Conference on Recommender Systems, pp. 20-23, Barcelona, España.

El objetivo del trabajo es adaptar las recomendaciones basadas en kNN por medio de la explotación exclusiva de ratings en el entorno temporal cercano del momento de recomendación. El aporte de este trabajo es el mismo logrado en (R3) y (C5).

1.4.3 RESUMEN

En la Figura 1-1 se muestra un gráfico que resume todas las contribuciones de los objetivos de investigación.

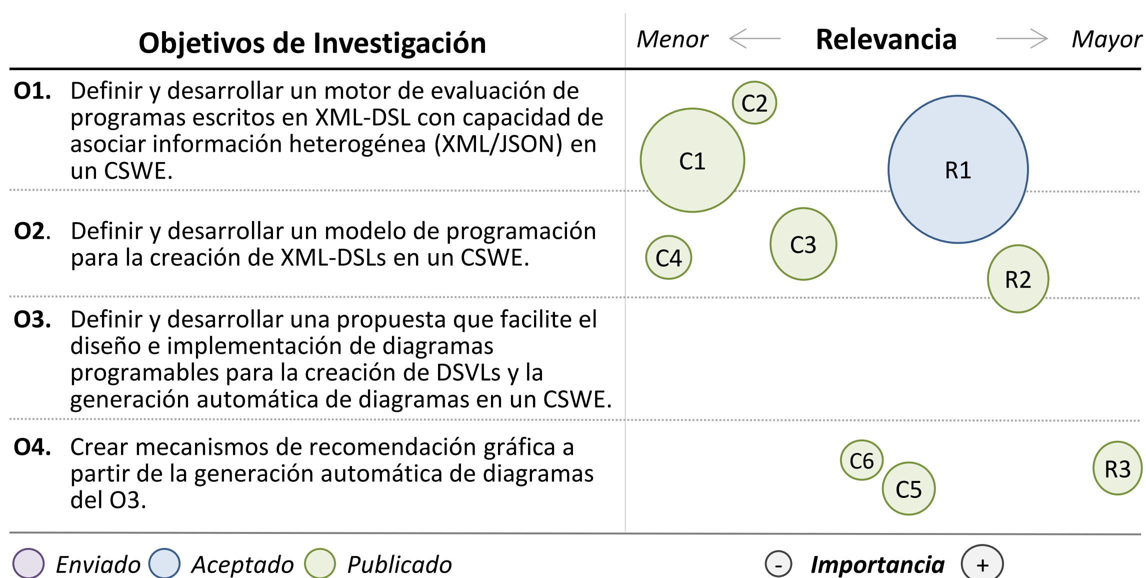


Figura 1-1. Distribución de las contribuciones de objetivos de investigación respecto a la importancia del aporte y su relevancia.

Cada círculo está referido a cada una de las diferentes contribuciones. Los colores indican el estado de la contribución (enviado, aceptado o publicado). El tamaño del círculo nos indica la importancia de la contribución respecto de los objetivos de investigación. Finalmente, de izquierda a derecha se indica la relevancia de la contribución.

1.5 ESTRUCTURA DE LA TESIS

Esta tesis está dividida en nueve capítulos:

1 El capítulo "INTRODUCCIÓN" introduce la motivación, el marco tecnológico, los objetivos de investigación y los aportes, transferencias y publicaciones, realizados en este trabajo de tesis.

2 El capítulo "ESTADO DEL ARTE" hace un resumen de los lenguajes de dominio específico, los intérpretes de programación y los lenguajes XML y su uso en un CSWE. Además, se detalla una revisión de las herramientas y entornos de desarrollo para la creación y evaluación de DSL y DSVL. Por último, se hace una revisión sistemática en revistas de la base de datos del Journal Citation Report de temas relacionados con intérpretes, DSL y DSVL.

3 El capítulo "PSIENGINE: MOTOR DE EVALUACIÓN PSI" expone el motor de evaluación de programas escritos con gramática XML en un CSWE (O1) el cual se denominará **Motor de Evaluación Psi** (denotado por **PsiEngine**). Se establecerán los elementos de programación JavaScript a utilizar. Seguido se formalizarán la construcción de gramáticas Psi, lenguajes Psi y componentes Psi dentro del PsiEngine. Además, se explicará en detalle el *Intérprete PsiXML*, pieza clave del PsiEngine y se expondrá una metodología para crear DSL basados en los lenguajes Psi. Como caso de estudio y ejemplo ilustrativo, se mostrará el lenguaje de programación MiniPsi.

4 El capítulo "PSIMODEL: MODELO DE PROGRAMACIÓN PSI PARA LA GENERACIÓN DE CÓDIGO JAVASCRIPT" establece un **Modelo de Programación** consistente en crear un conjunto de elementos de programación basados en lenguajes Psi en un CSWE, para generar código JavaScript y documentación (O2). Este modelo de programación se denominará **PsiModel** y se fundamenta en la programación orientada a objetos, en la construcción de widgets y componentes, y en la creación de lenguajes Psi interpretebles por el PsiEngine.

5 El capítulo "PSIDIAGRAM: DIAGRAMA PROGRAMABLE PSI" implementa el diagrama programable del O3, el cual se denominará **PsiDiagram**. El PsiDiagram es el encargado de la elaboración de modelos gráficos y diagramas, donde su apariencia y contenido depende de múltiples fuentes heterogéneas de información (XML y/o JSON), a partir de los conceptos línea, figura, lista y contenedor. Adicionalmente, se presentan funcionalidades y herramientas como cajas de diálogo, barras de herramientas, ventanas emergentes, menús, etc., que pueden ser asociadas al diagrama programable. Como parte de las herramientas se presenta el generador de diagramas a partir de fuentes de información y reglas de selección. Con este generador se construye el recomendador de diagramas, donde se crean los mecanismos de recomendación para la definición de DSVL basados en el diagrama programable (O4).

6 El capítulo "PSIENVIRONMENT: ENTORNO DE DESARROLLO PSI" describe e implementa el entorno de desarrollo **PsiEnvironment** basados en **PsiModel** y **PsiDiagram**. Este entorno tiene como objetivo facilitar la creación de widgets, componentes JavaScript, componentes Psi, diagramas Psi y diagramas programables Psi. Adicionalmente implementa varias métricas software estándar, para validar componentes, lenguajes Psi y diagramas programables Psi.

7 El capítulo "TRANSFERENCIA TECNOLÓGICA" implanta parte de los resultados de esta tesis en la Plataforma B2T SCOOP, mediante cuatro proyectos: Proyecto ADG (generación de diagramas y

modelos gráficos automáticos), Proyecto SID (Interfaces de diagrama SCOOP), Proyecto BPME (generación del estándar BPMN 2.0 con PsiDiagram) y Proyecto SMAPPS (creación de aplicaciones móviles basadas en BPMN 2.0 con PsiDiagram).

8 El capítulo "METODOLOGÍA DE VALIDACIÓN Y RESULTADOS" establece los resultados y la validación de esta tesis. Se divide en casos de estudio para validar el PsiEngine del Capítulo 3, lista de lenguajes Psi para validar el PsiModel del Capítulo 4 y casos de estudio, generación de diagramas y lenguajes visuales para validar el PsiDiagram del Capítulo 5. Como parte de la validación de todos ellos, se calcula y analizan las métricas software obtenidas a partir de PsiEnvironment. Por último se hace una síntesis de todos los resultados obtenidos.

9 Por último, el capítulo "CONCLUSIONES Y TRABAJOS FUTUROS" concluye la tesis con un resumen de las principales aportaciones y se exponen nuevas líneas de trabajo.

Capítulo 2. ESTADO DEL ARTE

Este capítulo está dedicado a la revisión de los trabajos relacionados con nuestra investigación. Se divide en cuatro secciones: *conceptos previos*, *los lenguajes*, *frameworks e IDEs* y *estudios realizados*. En la sección de *conceptos previos* se resume los conocimientos básicos necesarios para la consecución del objetivo de esta tesis. En la sección de *los lenguajes* se profundiza en los lenguajes XML, los DSLs y lenguajes visuales, haciendo énfasis sobre los DSLVs. En la sección de *frameworks e IDEs* se hace un resumen de los principales entornos de desarrollo integrado y entornos de modelado visual. La última sección, en *estudios realizados* se presenta un estudio sobre un conjunto de funcionalidades deseables para los entornos de modelado visual y el resumen de la *revisión sistemática* de esta investigación, enfocado hacia el estudio de los XML-DSLs e Intérpretes en el CSWE.

2.1 CONCEPTOS PREVIOS

El objetivo de esta sección es resumir el conocimiento sobre las aplicaciones web, la Web 2.0, los intérpretes de programación y algunos conceptos de programación necesarios.

2.1.1 ENTORNO CLIENTE WEB (CSWE) Y APLICACIONES WEB

El contexto general de trabajo en el que se desarrollo esta tesis, tal como se ha mencionado en el Capítulo 1, es el Entorno Cliente Web (CSWE). Este reúne el conjunto de *tecnologías*, *lenguajes*, *metodologías de programación* y *frameworks* disponibles en el cliente web para la implementación de componentes, widgets web o aplicaciones web (ver Figura 2-1, Cliente Web). Entre las *tecnologías* y *lenguajes* se incluyen básicamente: HTML5, CSS3, DOM, AJAX, JSON, JavaScript, XML, XSLT, XLink y SVG. Las *metodologías de programación* a usar son la programación orientada a componentes y uso de patrones de programación. Por último, los *frameworks* empleados facilitan la implementación de componentes en un cliente web, entre los más usadas en esta tesis mencionamos jQuery, AngularJS, jQuerUI, codemirror, entre otros.

En la actualidad, la creación de aplicaciones web se fundamenta en la creación de páginas web dinámicas, ya sea a nivel de servidor web o de cliente web. En la Figura 2-1 se muestra la arquitectura multinivel (*Capa de Presentación*, *Capa de Negocios* y *Capa de Datos*) y las principales tecnologías para la creación de aplicaciones web.

En la creación de páginas web dinámicas, el lenguaje HTML establece la distribución de contenido, el lenguaje CSS precisa el estilo y presentación (W3C: HTML & CSS, 2010), y los lenguajes de scripting (W3C: JavaScript Web APIs, 2010) y el JavaScript Asincrónico y XML AJAX, define la funcionalidad y el

dinamismo de la página (Holzner, 2006; Jingjing & Chunlin, 2012). Adicionalmente, en una página web se pueden incluir imágenes, video y/o audio usando tecnologías como PNG, JPEG y AVI (W3C, 2014; McDaniel, 2011; Crane, et al., 2005).

A nivel de servidor web, existe un abanico de tecnologías para crear páginas dinámicas dependiendo del lenguaje de programación (ver Figura 2-1, Servidor Web), entre la que se pueden mencionar Java, C#, PHP, Perl y Python.

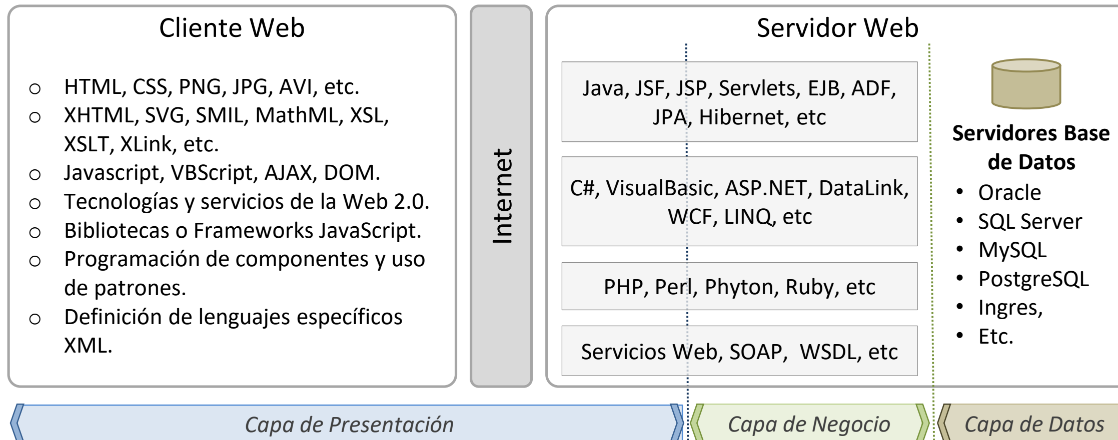


Figura 2-1. Arquitectura multinivel para la creación de aplicaciones web.

Un lenguaje de scripting (JavaScript, JScript, o VBScript) de un navegador, es un lenguaje de programación interpretado, imperativo, orientado a objetos y basado en prototipos. Se rige bajo la especificación ECMAScript (ECMA, 2011). En la programación basada en prototipos, las "clases" no están presentes, se obtienen a través de la clonación de objetos ya existentes que sirven de prototipos, y su herencia se adquiere mediante la extensión de su funcionalidad. Al incluir AJAX (Crane, et al., 2005), se permite comunicación asíncrona con el servidor posibilitando la modificación de páginas web, sin necesidad de recargarlas, mejorando la velocidad, interactividad y usabilidad de la aplicación web. Por otro lado, los lenguajes de scripting incluyen APIs que implementan el modelo de objetos de documento (DOM, Document Object Model (W3C, 2004)) para facilitar la navegación por el contenido y la estructura de un documento HTML o XML, permitiendo modificar, ocultar, mover, eliminar y/o cambiar dinámicamente las características del documento.

En la actualidad, los clientes web cuentan con recursos de procesamiento muy rápidos y se encuentran en diferentes dispositivos (ordenadores, móviles, tabletas o televisión). Así mismo hay una gran diversidad de tecnologías, bibliotecas y *frameworks* para la construcción de páginas web dinámicas directamente en el navegador. Dentro de un CSWE, surgen diversos *frameworks* JavaScript basados en MVC (Modelo Vista Controlador) que ayudan a implementar estás páginas. En (Osmani, 2012) se muestra un análisis comparativo entre diversos *frameworks* JavaScript MVC como: Backbone (backbonejs.org), Ember (emberjs.com), Angular (angularjs.org), Dojo (dojotoolkit.org), YUI (yuilibrary.com), Batman (batmanjs.org), entre otros. Su objetivo principal es implementar una misma aplicación web **TodoMVC** (Osmani, et al., 2012) en los diferentes frameworks para ayudar a simplificar el estudio y entendimiento de cada uno de ellas. Por ejemplo, AngularJS (Google, 2010) amplía el HTML tradicional con etiquetas y atributos adicionales que permiten la sincronización automática de los modelos y las vistas. Backbone (Ashkenas, 2010) estructura las aplicaciones web basándose en el paradigma de diseño MVP (Modelo Vista Presentador). Dojo (Dojo Fundation, 2007) tiene una API y una variedad de widgets que facilitan la creación de aplicaciones Web. Polymer (Bradley, 2013) se basa en la

creación de componentes web en forma declarativa, permitiendo enlaces de datos bidireccionales, manejo de eventos y herencia declarativa.

2.1.2 LA WEB 2.0

El término Web 2.0 se puede considerar como "*todas aquellas utilidades y servicios de Internet que se sustentan en una base de datos, la cual puede ser modificada por los usuarios del servicio, ya sea en su contenido (añadiendo, cambiando o borrando información o asociando metadatos a la información existente), bien en la forma de presentarlos o en contenido y forma simultáneamente*" según (Ribes, 2007).

La Web 2.0 (Marín de la Iglesia, 2010) ofrece tecnologías, servicios y herramientas necesarias en el cliente web para construir páginas verdaderamente funcionales, agradables y usables. Dentro del conjunto de tecnologías y servicios se puede enumerar: AJAX**,** Mashup *Remezcla* (Holdener III, 2007), *Permalinks* (Van Duyne, et al., 2006)**,** REST *Representational State Transfer* (Costello, 2005)*,* OPML *Outline Processor Markup Language* (OPML, 2007), P2P *Peer-to-Peer* (Kolweyh & Lechner, 2006), RSS *Really Simple Syndication* (RSS Specifications, 2007) *y* Ruby on Rails (Matsumoto, 2008).

En la Web 2.0 los usuarios gestionan la información según sus necesidades: pueden agregar, modificar y/o borrar información. Además, la Web 2.0 posibilita el diseño de las aplicaciones web para diferentes dispositivos y navegadores, y crea un conjunto de servicios y contenidos susceptibles de ser transformados, mezclados y/o modificados para su visualización. Con el uso de la programación extrema (Fredrick, 2003), el conjunto de servicios y contenidos y la arquitectura colaborativa, se puede plantear cualquier modelo de negocio y dar solución a un dominio específico.

Una herramienta útil en el desarrollo de un sistema de información es el Widget[3] (Northover & Wilson, 2004). El Widget es una pequeña aplicación que tiene acceso a información y que se visualiza a través de una ventana, ya sea en el escritorio de un ordenador, en una página web, en una tableta, o en un móvil. Existen widgets de distribución gratuita a través de Internet, y son relativamente fáciles desarrollar en cualquier lenguaje de programación. En la Web 2.0, un widget para una página web se denomina Widget Web y se implementan con distintos lenguajes como: HTML, CSS, JavaScript, Flash, Applet, Silverlight y/o Windows Media Player entre otros. Su objetivo es enriquecer los contenidos y funcionalidades de la aplicación Web. En (Northover & Wilson, 2004) se encuentra la especificación que estandariza un widget en un cliente web.

2.1.3 LOS INTÉRPRETES DE PROGRAMACIÓN

Un **intérprete** es una aplicación que toma un *Programa* **P** escrito en un *Lenguaje Fuente* **LF** (en adelante **P**/**LF**) y uno o más datos, los analiza y evalúa simultáneamente para obtener unos resultados (ver Figura 2-2). Los compiladores, a diferencia de los intérpretes, transforman el **P**/**LF** a código objeto (proceso de compilación), con él lo ejecutan, toman los datos de entrada y generan los resultados (Ruiz, 2010; Alfonseca, et al., 2006).

En la Figura 2-2 se muestra el proceso y componentes principales que intervienen en un intérprete. El **Traductor** hace un primer proceso de *Scanner*, consistente en la descomposición del programa fuente en componentes léxicos (Análisis Léxico). Posteriormente el proceso de *Parsing* agrupa estos componentes en frases gramaticales (Análisis Sintáctico) y comprueba la validez semántica de las sentencias aceptadas por el lenguaje (Análisis Semántico). El resultado del traductor es la *Representación Interna* RI (o código intermedio) del *Lenguaje Fuente*. En caso de que existan errores de algún tipo, estos

[3] Un **Widget** es considerado también como la abreviación de las palabras window-gadget, que se interpretaría como aparato, herramienta, artilugio o dispositivo dentro de una ventana.

se proporcionan como salida. Durante el proceso se genera la *Tabla de Símbolos*, que contiene información asociada al lenguaje (Mozgovoy, 2010).

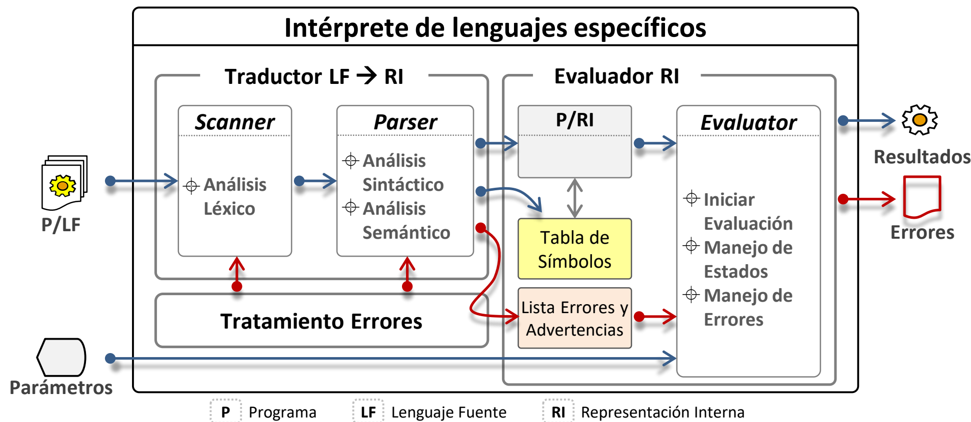


Figura 2-2. Intérprete de un *Programa* P escrito en un *Lenguaje Fuente* P/LF.

En el intérprete, el **Evaluador** es el encargado de tomar la *Representación Interna* RI y los datos de entrada, y llevar a cabo las acciones indicadas de evaluación del programa para obtener los resultados en la salida. Durante este proceso se deben contemplar los errores de evaluación.

Los Intérpretes por lo general son sencillos de implementar aprovechando herramientas como Yacc, Bison y entornos de desarrollo integrado (ver sección 2.3). Por otro lado, los lenguajes interpretados, al poderse modificar y/o ampliar el lenguaje fuente a medida que se evalúa, proporcionan una mayor flexibilidad. Si el intérprete es portable a distintas plataformas se posibilita el incremento de su uso.

2.1.4 PATRONES Y DESARROLLO BASADO EN COMPONENTES

El concepto de patrón fue utilizado inicialmente por la ingeniería civil y la arquitectura, y posteriormente fue adoptado por la Ingeniería de Software. En (Gamma, et al., 2003) popularizó el concepto «*cada patrón es una regla de tres partes, expresa una relación entre un cierto contexto, un problema y una solución*». En (Bushmann, 1996) «*los patrones actúan como bloques de construcción para construir diseños más complejos. Este método de usar artefactos de diseño predefinidos permite mejorar la velocidad y calidad de su diseño. Los patrones ayudan a resolver problemas pero no proveen soluciones completas*». Según el grado de abstracción, existen diversos patrones para todo el ciclo de desarrollo de software, en (Berrospi, 1990; Bushmann, 1996; Deepak, et al., 2003; Ramirez, 2004) proponen patrones de diseño, de arquitectura, de creación, de comportamiento, de modelaje de requerimientos y de programación, entro otros. Los patrones de programación, también conocidos como *modismos* o *estilos de programación* ("Idioms" en inglés) son patrones de bajo nivel de abstracción, describen cómo implementar ciertas tareas usando un lenguaje de programación concreto, implican el desarrollo de partes de componentes o de relaciones entre ellos usando las características del lenguaje (Bushmann, 1996).

Un **componente** son fragmentos de código que se adhieren a un programa en tiempo de ejecución o compilación (Berrospi, 1990). Cuando se usa la reutilización de la Programación Orientada a Objetos[4] (OOP, Object-oriented programming) en la construcción de un componente, el componente se denomina

[4] La Programación Orientada a Objetos (OOP, Object-oriented programming) es un paradigma de programación basado en el uso de objetos (entidad que dispone de un estado y un comportamiento) para implementar programas informáticos. Incluyen conceptos de herencia, abstracción, polimorfismo, encapsulamiento, cohesión, acoplamiento, componente, biblioteca, librería, patrón entre otros (Bruegge & Dutoit, 2002)

componente reutilizable y se puede definir como «*cualquier componente que es específicamente desarrollado para ser utilizado, y es actualmente utilizado, en más de un contexto*» (Karlsson, 1995). Para que un componente sea reutilizable debe cumplir como mínimo: ser identificable, accesible solo a través de su interfaz, sus servicios predefinidos, documentado, y ser capaz de comunicarse con otros componentes. Es deseable que el componente cumpla con las características de: lo más genérico posible, auto-contenido, mantenido, certificado e independiente de la plataforma, del lenguaje de programación y de las herramientas de desarrollo (Montilva, et al., 2003).

En la ingeniería del software, el paradigma del **desarrollo de software basado en componentes** busca al igual que otras formas de abstracción, ventajas en el proceso de análisis, diseño e implementación, repercutiendo en ahorros de costes. Se considera que muchas características de los componentes parten del concepto de objetos, pero más allá del campo del software, el concepto viene de la industria en general, de la necesidad de estandarizar las partes de productos elaborados en línea, como los automóviles, electrodomésticos, etc. (Ariza & Molina, 2004). El objetivo final en Ingeniería de Software es contar con un mercado global de componentes, igual que ocurre con el hardware o con otras ingenierías.

2.2 Los Lenguajes

Esta sección resume la importancia de los lenguajes XML para la creación de aplicaciones web, y la importancia de los DSLs, XML-DSLs y los DSVLs en la Ingeniería de Software.

2.2.1 Plataforma Web Abierta y los lenguajes XML

Los estándares de la W3C (w3.org/standards) definen una *Plataforma Web Abierta* que permitan a los desarrolladores crear aplicaciones web disponibles en cualquier dispositivo (ordenadores, tabletas, televisión, móviles, e incluso automóviles). Su piedra angular es el HTML5, pero todo el poder de la plataforma se basa en muchas tecnologías que la W3C y sus socios están creando, incluyendo lenguajes de diseño web, web semántica, lenguajes XML y una variedad de APIs. Las aplicaciones web implican un conjunto de estándares para construcción y renderización de páginas web, en las que se incluyen CSS, PNG, AVI, AJAX, DOM, y los lenguajes XML como XHTML (W3C: XHTML, 2000), XSL (W3C: XSL, 2007), MathML (W3C: MathML, 2003), SVG (W3C, 2011) y/o SMIL (W3C: SMIL 3.0, 2008).

El lenguaje XHTML, es la versión XML del lenguaje HTML. Una de las características especialmente útiles del mismo, es la posibilidad de incorporar otros lenguajes XML con sus definiciones de espacios de nombres, para enriquecer el modelo de presentación de una página web. Por ejemplo: el lenguaje SMIL permite integrar audio, video, imágenes, texto o cualquier otro contenido multimedia para describir el comportamiento de su presentación multimedia; el lenguaje SVG sirve para crear gráficos vectoriales bidimensionales, tanto estáticos como animados; y, el lenguaje MathML tiene como objetivo expresar notación científica y mostrar documentos con expresiones matemáticas en páginas web.

El lenguaje XML (W3C: XML, 2008) es un estándar para el intercambio de información estructurada, entre diferentes programas. Esta información se puede transformar y formatear empleando la familia de lenguajes XSL. El lenguaje XSLT (W3C: Transformation, 2010) es el lenguaje que se usa para la transformación y requiere del lenguaje XPath (W3C: XPath, 2007) para la búsqueda y obtención de información. El lenguaje XSL-FO (W3C: Transformation, 2010), permite crear documentos en formato PDF a partir de la información XML. Con la ayuda del lenguaje XLink (W3C: XLink, 2001), se puede acceder a otras fuentes de información como documentos, imágenes y/o ficheros de Internet, para ampliar su contenido. Por último, XQuery (W3C: XQuery, 2007) es un lenguaje de consultas diseñado para buscar en colecciones de datos XML.

Los servicios web facilitan la comunicación entre aplicaciones basados en los estándares como SOAP y WSDL. Así mismo, la W3C establece estándares y tecnologías para la web semántica (RDF, SPARQL, OWL y SKOS), la arquitectura base (URI y HTTP) y el acceso web en cualquier lugar, momento y dispositivo.

En resumen, todos los estándares y borradores de la W3C útiles para la construcción de aplicaciones web se encuentran organizados en la página w3.org/TR.

2.2.2 LENGUAJES DE DOMINIO ESPECÍFICO

El término "**Lenguaje de Dominio Específico**" (DSL, Domain Specific Language) no está rigurosamente definido en la literatura: (Fowler, 2010) lo define como «*domain-specific language (noun): a computer programming language of limited expressiveness focused on a particular domain*»; (Spinellis, 2001) dice «*DSL is a programming language tailored specifically to an application domain: rather than being for a general purpose, it captures precisely the domain's semantics*»; y (Mernik, et al., 2005) lo define como *«domain-specific languages (DSLs) are languages tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use compared with general-purpose programming languages in their domain of application»*. Todos ellos confluyen en que un DSL es un lenguaje de programación que está dirigido a un problema específico, tal que su sintaxis y semántica contienen el mismo nivel de abstracción que ofrece el dominio del problema, y su objetivo es facilitar el diseño, definición e implementación de sistemas de información que den solución al problema del dominio. Además, el DSL proporciona las primitivas adecuadas que permiten a los expertos del dominio realizar esas tareas de manera más eficiente y producir sistemas de mayor calidad y fiabilidad.

Un *modelo* es la abstracción formal de un dominio específico. En otras palabras, un *modelo* es un conjunto de elementos formales que describen ese dominio, o parte de él. Distintos modelos pueden definir distintos componentes y funcionalidades que conforman el dominio (Mellor, et al., 2003; Schmidt, 2006). Un *metamodelo* es un modelo que proporciona la base para la construcción de otro modelo. Aunque ambos son modelos, uno se expresa en términos del otro, es decir, el modelo es una instancia del metamodelo.

La gramática de un DSL es definida mediante un metamodelo, comúnmente conocida como la *sintaxis abstracta* del DSL. La *sintaxis abstracta* se refiere a la sintaxis del metamodelo, al cual hay que crear una *sintaxis concreta* en forma de notación de texto (sintaxis textual concreta) o de diagrama (sintaxis gráfica concreta). Una sintaxis textual concreta permite a los usuarios trabajar como si fueran lenguajes de programación basados en textos, que a su vez se dividen en textos con gramática general o textos con gramática XML (XML-DSL). Mientras que una sintaxis gráfica concreta permite a los usuarios trabajar médiate lenguaje de programación visual, denominado Lenguaje Visual (Kats, et al., 2009; Fowler & White, 2010; Kleppe, 2008). El lenguaje visual más popular es UML. Los lenguajes visuales se resumirán en la siguiente sección.

La creación de un DSL requiere crear un intérprete o compilador, el cual toma programas escritos en el lenguaje fuente DSL, datos y recursos, los analiza y los evalúa, para obtener los resultados o soluciones. Un compilador, a diferencia del intérprete, transforma el código fuente a código objeto (proceso de compilación), el cual se ejecuta tomando ciertos datos de entrada y generando los resultados (Ruiz, 2010; Alfonseca, et al., 2006). Los intérpretes y compiladores suelen implementarsen aprovechando herramientas como Lex & Yacc (Brown, et al., 1992) o Flex & Bison (Levine, 2009), y su uso es extensivo a nivel de aplicaciones de escritorio y servidores. Los IDEs como Eclipse, Visual Studio y ASF+SDF Meta-Environment también ofrecen herramientas y lenguajes para la implementación de DSLs.

2.2.3 LENGUAJES VISUALES

Los **Lenguajes Visuales** (VL, Visual Languages) son representaciones gráficas que han sido utilizadas en la ingeniería del software para la comprensión y construcción de sistemas de información. Dependiendo del contexto de la aplicación, se clasifican en Lenguajes Visuales de Propósito General (GPVL, General Purpose Visual Languages) y Lenguajes Visuales de Dominio Específico (DSVL, Domain Specific Visual Language).

Los GPVLs proporcionan elementos gráficos para modelar potencialmente cualquier sistema. El más difundido es el Lenguaje de Modelado Unificado (UML, Unified Modeling Language) y sirve para visualizar, especificar, construir y documentar un sistema de información (OMG, 2009; Pender, 2003; Miles & Hamilton, 2006).

Los DSVLs son representaciones gráficas que permiten definir un conjunto de elementos gráficos que describen formalmente los conceptos de un dominio o parte del él. Un diseño apropiado permite que personas sin conocimientos de programación, pero con conocimientos del dominio puedan desarrollar nuevas aplicaciones en ese dominio (Kelly & Tolvanen, 2008). En esta tesis nos centraremos en la creación y uso de DSVLs.

Existen diversos trabajos y herramientas para definir e implementar DSVL. En (Guerra, 2007) se propone una herramienta de metamodelado que es usada para generar entornos de DSVL en diversos dominios. De manera similar, en (Bottini & Grau, 2004; Bottini, et al., 2010) se especifica un conjunto de metamodelos para la definición y gestión de lenguajes visuales. En (Heer & Agrawala, 2006) se propone un conjunto de patrones de diseño de software para la visualización de modelos de datos estructurados. Además, en la Ingeniería Dirigida por Modelos (MDE[5], Model Driven Engineering) se estableció la necesidad de lenguajes de metamodelado de dominio específico (DSMM) ofreciendo elementos formales o primitivas orientados a la definición de lenguajes de modelados para un dominio específico (de Lara, et al., 2013; de Lara & Guerra, 2012). Otros planteamientos interesantes para definir e implementar DSVL son ATL Tools (Jouault, et al., 2008) o transML (Guerra, et al., 2013).

A lo largo del tiempo se han ido desarrollando algunos DSVLs. Como ejemplos se pueden mencionar el Lenguaje de Modelado Web (WebML, Web Modeling Language), el modelo de base de datos y el Modelo de Procesos de Negocios (BPMN, Business Process Modeling Notation). El WebML ofrece una notación visual para el diseño de aplicaciones Web (Mishra & Mohanty, 2011). El modelo de base de datos determina la estructura lógica y el modo de almacenar, organizar y manipular los datos, donde el más usado es el diagrama entidad-relación (Teorey, et al., 2011). El BPMN es un estándar que permite el modelado de procesos de negocio basado en el concepto de flujo de trabajo (Pant & Juric, 2008; OMG, 2011).

2.3 FRAMEWORKS Y ENTORNOS DE DESARROLLO INTEGRADO PARA LA CREACIÓN DE DSLS Y DVSLS

La creación de un DSL requiere la implementación de un intérprete o compilador para especificar la gramática del lenguaje y la definición de su funcionamiento. Usualmente, suelen ser creados aprovechando herramientas como Lex & Yacc o Flex & Bison, pero los Entornos de Desarrollo Integrado (IDE, Integrated Development Environment) como Eclipse, Visual Studio y ASF+SDF Meta-Environment ofrecen herramientas y lenguajes para su implementación. Adicionalmente, los IDEs ofrecen herramientas para la construcción de DSVLs.

[5] La MDE combina los conceptos de DSLs, XML-DSLs y/o DSVLs con motores de transformación de modelos y generación de código.

2.3.1 FRAMEWORKS PARA LA CREACIÓN DE DSLS

En la actualidad, los lenguajes de propósito general más utilizados para la construcción de DSLs son: Java (Kats, et al., 2009; Parr, 2013; Granback, 2009; Bettini, 2013), C# (Kourie, et al., 2008), Haskell (Serrano, 2014), Groovy (Dearle, 2010; Ghosh, 2010), Clojure (Kelker, 2013) y Python (McGuire, 2007). Estos lenguajes son utilizados en diversos IDEs que ofrecen *plugins* o *frameworks* para la creación de DSLs, XML-DSLs y/o DSVLs:

- **Eclipse**

En Eclipse se puede mencionar los frameworks para la creación de DSL como: Stratego/XT, LISA, Spoofax, Antlr, Xtext y Eclipse Modeling Project. Stratego/XT es un lenguaje y un conjunto de herramientas para la construcción de sistemas de transformación de programas independientes (Bravenboer, et al., 2008). LISA es un entorno visual interactivo para el desarrollo de lenguajes de programación (Mernik, et al., 2002). Spoofax es un framework para crear DSL textuales basada en ADF (ADF, Algebraic Specification Formalism) (Kats, et al., 2009). Antlr permite crear analizadores, intérpretes, compiladores y traductores de lenguajes basados en algoritmos LL(*) (Parr, 2013). Xtext permite crear programa de análisis, generador de código, intérprete, entre otras (Bettini, 2013; Bettini, et al., 2013). Eclipse Modeling Project combina los DSL textuales y gráficos con el desarrollo dirigido por modelos (MDD, Model-Driven Develepment) para mejorar la productividad, la calidad, y aislar los sistemas de información de los continuos cambios tecnológicos (Granback, 2009).

- **Visual Studio**

En .NET se pueden mencionar: DSL Models y Boo. DSL Models proporciona la tecnología para para crear DSL textuales y gráficos (Cook, et al., 2007). Mientras que Boo es un lenguaje especializada para la creación de DSL textuales (Rahien, 2010).

- **ASF+SDF Meta-Environment**

El ASF+SDF Meta-Environment es un IDE que contiene un conjunto de herramientas para el análisis de programación interactiva y transformación. Combina SDF (Sintaxis Definición formalismo), ASF y otras tecnologías para la definición e implementación de DSLs generando código en C de alto rendimiento (Van Den Brand, et al., 2002).

- **Rascal**

Rascal es un DSL experimental para metaprogramación, se fundamenta en la integración de análisis de código estático[6] y la manipulación a nivel conceptual, sintáctico, semántico y técnica para la creación de DSL. Su sintaxis y semántica se basan en procedimientos (imperativos) y la programación funcional (Klint, et al., 2013).

- **Cliente web**

Al reducir el área de trabajo para la creación de DSLs a un cliente web y sus limitaciones, tenemos soluciones como *Jison*. *Jison* crea analizadores sintácticos ascendentes en JavaScript para DSL textuales (Carter, 2009). Como ejemplos ilustrativos DSL generados con *Jison* están: CoffeScript (Lee, 2014), JSONLint (Carter, 2012), Orderly (Carter, 2009) y js-sequence-diagrams (Bramptom, 2012). Por otro lado, en esta área de trabajo, para gestionar DSLs basadas en gramáticas XML, se usa DOM y JavaScript y su programación se hace ad-doc. Otra solución, es usar las características del navegador para crear plugines y generar DSLs, limitado a un tipo de navegador.

[6] Es el análisis de los programas informáticos sin llegar a su ejecución, es decir, hace un análisis del código fuente o del código objeto.

2.3.2 Herramientas para la creación de Diagramas y Lenguajes Visuales en CSWE

Como quedó definido anteriormente, un **modelo** es una representación abstracta, conceptual o gráfica de procesos o problemas de dominio específico, con el fin de analizarlos, describirlos, explicarlos y/o solucionarlos. Distintos modelos pueden definir distintos componentes y funcionalidades que conforman el dominio (Mellor, et al., 2003; Schmidt, 2006). Un caso particular son los **modelos gráficos**, los cuales se sirven de representaciones gráficas para describir el conocimiento, pero si este modelo gráfico es utilizado en la Ingeniería de Software para la comprensión y construcción de sistemas de información, se denomina **lenguaje visual**.

En general, un **diagrama** es un dibujo geométrico y abstracto que representa la información de un problema, fenómeno, proposición o sistema. El concepto de diagrama en esta tesis, incluye como sinónimos modelos gráficos, lenguajes visuales, bocetos, pinturas, gráficos, etc. En la actualidad, se usa una gran variedad de diagramas para visualizar problemas específicos, por ejemplo diagramas: de estados, de bloques, de clases, de Gantt, de Venn, PERT, de procesos de negocios, de colaboración, en árbol, etc.

La Web 2.0 (Marín de la Iglesia, 2010; Anderson, 2012), aporta gran variedad de tecnologías, servicios y herramientas para la construcción de páginas web funcionales, agradables y usables, con la posibilidad de desplegarse en diferentes tipos de dispositivos. En este ámbito, la web 2.0 ofrece *widgets*, *framewoks* JavaScript y aplicaciones web para la creación de diagramas enfocados a la **visualización y análisis de datos** que enriquecen la presentación de la información. Puede mencionarse Highcharts (highcharts.com), D3.js (d3js.org), FusionCharts (fusioncharts.com), Graphviz (graphviz.org), Google Charts (developers.google.com/chart), iCharts (icharts.net), Excel Analyze (office.live.com), Protovis (mbostock.github.io/protovis), entre otros. Estas herramientas son vistosas, fáciles de usar, y se basan en información JSON con una estructura predefinida.

Por otro lado, se encuentran las aplicaciones web cuyo objetivo son la **creación de diagramas** en línea, entre las que se destacan Gliffy (gliffy.com), LucidChar (lucidchart.com), yFiles (yworks.com), GoJS (nwoods.com) y Cacoo (cacoo.com). Estas se basan en manipular y administrar documentos SVG, canvas HTML o Adobe Flash, principalmente y usan librerías o plantillas gráficas para su fácil edición, e implementan herramientas para la interacción y modificación del diagrama y sus elementos gráficos. En esta misma línea, existe *frameworks* que ayudan a facilitar la creación de diagramas basados en SVG como: *Raphaël* (raphaeljs.com), *jQuerySVG* (keith-wood.name/svg.html), JSPlumb (jsplumb.org) y JointJS (jointjs.com). Por ejemplo, *Raphaël* simplifica el manejo de los gráficos vectoriales y proporciona un lenguaje que es compatible con los diferentes navegadores; mientras que *jQuerySVG* implementa una interacción con el lienzo SVG y creación de elementos gráficos basados en jQuery.

2.4 Identificación de Funcionalidades para la creación de diagramas y lenguajes visuales

En la Tabla 2-1 se presenta un conjunto de funcionalidades deseables para **creación de diagramas** en línea para un CSWE. Se incluye la necesidad de asociar herramientas visuales como: menú flotante, ventana emergente, caja de diálogo, barra de herramientas, entre otros. Además, combina con el uso de información heterogénea (XML/JSON) para la elaboración de los diagramas.

En la Tabla 2-2 se muestra una comparación de las herramientas existentes y cuál de estas funcionalidades de la Tabla 2-1 están cubiertas. Adicionalmente, la tabla se divide entre las herramientas para entornos Web y las herramientas de escritorio mencionadas con anterioridad. Para cada una de ellas se validó con la documentación disponible, y en algunos casos se experimentó haciendo uso de ellas (gliffy, LucidChar, cacoo y VSDSK).

Tabla 2-1. Funcionalidades para la gestión de diagramas, asociación de herramientas visuales y asociación de información heterogénea.

Funcionalidad	Entorno de desarrollo	Resultados obtenidos
1. Gestión de diagramas	Crea, edita y manipula diagramas. En particular, define la sintaxis y la semántica visual abstracta y concreta de los elementos de un lenguaje visual.	Manipular el diagrama y sus elementos gráficos.
2. Herramientas de interacción	Contiene herramientas para la interacción y modificación tanto visual como de comportamiento del diagrama y sus elementos.	Facilitar la interacción con los diagramas y sus elementos gráficos.
3. Serialización de diagramas	Define reglas y mecanismos para almacenar el diagrama.	Almacenar el diagrama.
4. Renderización de diagramas	Define algoritmos de distribución de elementos gráficos para su representación de forma automática, teniendo en cuenta patrones para la visualización de información, adaptados en cada caso al tipo de información y al usuario final.	Obtener diferentes visualizaciones para un mismo modelo o representación gráfica.
5. Integración de fuentes de modelos heterogéneos	Define estrategias y reglas para la integración de modelos de fuentes heterogéneas, modelos escritos en diferentes lenguajes.	Obtener la visualización, en un mismo diagrama, de un conjunto de modelos de diversas fuentes de modelos gráficos o lenguajes visuales.
6. Asociación de fuentes de información heterogéneas	Define las reglas de asociación de fuentes de información heterogénea disponibles en el cliente web a los elementos del diagrama.	Obtener visualización o información heterogénea adicional sobre un diagrama.
7. Herramientas visuales para diagramas	Contiene herramientas como: menú flotante, ventana emergente, caja de diálogo, barra de herramientas, widgets, arrastrar y soltar, etc., para manipular y visualizar la información asociada de los modelos serializados.	Facilitar el acceso y la visualización de la información asociada a un modelo serializado. Modificar el comportamien-to de visualización según esta información.
8. Recomendación de diagramas	Mecanismos de recomendación de diagramas basados en la generación automática diagramas.	Facilitar al usuario la visualización de diagramas.

En términos generales, las herramientas de gestión de diagramas y de interacción están muy elaboradas e integran interfaces de usuario agradables y usables. No todas aportan renderización de diagramas, modelos gráficos y lenguajes visuales. La asociación de fuentes de información heterogénea (XML/JSON) no es primordial para estas herramientas.

Las herramientas para la construcción de diagramas y DSVLs como GMF/Graphit, VSSDK y VLDesk requieren de implementación ad-hoc para la inclusión de elementos visuales. En esta tesis, una de las metas es poder disponer fácilmente de herramientas para la creación de diagramas programables a nivel de aplicación web.

Por último, en las herramientas analizadas no se contempla la funcionalidad de recomendación de diagramas, que constituye un requisito interesante a incluir en la creación de modelos gráficos.

Tabla 2-2. Análisis comparativo de herramientas para la gestión de modelos.

Funcionalidad	gliffy	LucidChar	yFiles	GoJS	graphviz	cacoo	GMF/ Graphit	VSSDK	VLDesk
1. Gestión de diagramas	☑	☑	☑	☑	☑	☑	☑	☑	☑
2. Herramientas de interacción	☑	☑	☑	☐	☑	☑	☑	☑	☑
3. Serialización de diagramas	☑	☑	☑	☑	☑	☑	☑	☑	☑
4. Renderización de diagramas	☑	☑	☑	☑	☑	☑	☑	☑	☑
5. Integración de fuentes de modelos heterogéneos	☐	☐	☐	☐	☐	☐	☑	☑	☐
6. Asociación de fuentes de información heterogéneas	☐	☐	☐	☐	☐	☐	☑	☑	☑
7. Herramientas visuales para diagramas	☐	☐	☐	☐	☐	☐	☑	☑	☑
8. Recomendación de diagramas	☐	☐	☐	☐	☐	☐	☐	☐	☐

2.5 Mapeo Sistemático sobre DSLs en CSWE

De acuerdo con (Dearle, 2010; Fowler, 2010; Ghosh, 2010; Hudak, 1996; Kelker, 2013; Kosar, et al., 2010; Mernik, et al., 2005), los DSLs se convierten en una área de investigación dentro del campo de la Ingeniería de Software y uno de los pilares dentro de la metodología de desarrollo de software, tales como las fábricas de software (Greenfield & Short, 2004), la programación generativa (Czarnecki & Eisenecker, 2000), lenguajes orientados a la programación (Ward, 1995) o la ingeniería dirigida por modelos (MDE) (Schmidt, 2006).

En esta sección se revisan y resumen las conclusiones más relevantes que pueden extraersen de los estudios de mapeos sistemáticos sobre DSLs. A continuación, se establecen las tendencias de investigación de los XML-DSLs en CSWE con un estudio propio de mapeo sistemático.

2.5.1 Revisiones previas

En (Kosar, et al., 2016) se hace un Estudio de Mapeo Sistemático (SMS, Sistematic Mapping Study) sobre el área de investigación DSL para identificar las tendencias de investigación, los posibles temas abiertos y la demografía de la literatura. El SMSK se basa en la búsqueda sistemática de estudios primarios de revistas, conferencias y talleres durante el período de 2006 hasta el 2012, y sobre una encuesta realizada en (Mernik, et al., 2005).

Las preguntas de investigación de SMS de (Kosar, et al., 2016) permitió por una parte, determinar el *tipo de contribución* (técnicas/métodos, herramientas, procesos y mediciones), el *tipo de investigación* (empíricos, no empíricos, experimental, etc.) y las *fases de desarrollo* (análisis de dominio, diseño, implementación, validación y mantenimiento) de los DSLs en el período 2006-2012. Por otro lado, permitió establecer el *número de publicaciones anuales*, *top de documentos citados*, *instituciones de investigación* y *lugares de publicación* (revistas, conferencias o talleres).

- **Forma de contribución**

En (Kosar, et al., 2016) se estableció que la comunidad DSL se interesa más en el desarrollo de nuevas técnicas/métodos (79,3%) que apoyan a las diferentes fases de desarrollo DSLs o en la integración de los DSLs en los procesos de Ingeniería de Software (10,5%), en lugar de invertir en el desarrollo de nuevas herramientas de DSLs (6,9%) o en la medición de eficiencia de DSLs (3,3%).

En este aspecto, nuestra investigación se centra en desarrollar nuevas herramientas de DSLs, y en particular en el desarrollo de nuevas herramientas de XML-DSLs. Para ello se implementó un intérprete para evaluar XML-DSLs (O1) y se define un modelo de programación y un ambiente de desarrollo ligero para implementar estos XML-DSLs (O2) en un CSWE. La creación de herramientas XML-DSLs es un área poco explorada y completamente innovadora.

- **Fases de desarrollo**

En (Kosar, et al., 2016) determinó que las fases de desarrollo de un DSL, en su mayoría incluyen un análisis de dominio (58,7%), un diseño (74,6%) y una implementación (71,8%), mientras que la validación y el mantenimiento, solo el 3,9% de los estudios lo incluye. Es un resultado natural, ya que la mayoría de estudios primarios están interesados en el desarrollo de nuevas técnicas/métodos.

En particular, los trabajos previos de la sección 1.2.4 están dentro del desarrollo de nuevas técnicas/métodos. Pero el desarrollo de un motor de evaluación, la definición de un modelo de programación y un ambiente de desarrollo ligero para implementar en un CSWE, deben incluir todas las fases de desarrollo de un DSL y debe establecer una metodología formal para la creación de XML-DSLs.

- **Herramientas y lenguajes de programación**

En (Mernik, et al., 2005) se identificaron más de 20 herramientas de desarrollo para la creación de DSLs, incluidas las mencionadas en la sección 2.3.1. En esta tesis, se hace hincapié en la creación de una aplicación web que sirva como entorno ligero de desarrollo de XML-DSLs, es decir se excluyen IDEs como Visual Studio, Eclipse o Netbeans. Aunque es cierto que se puede pensar en un mejor entorno de desarrollo de XML-DSLs basados en estos IDEs, el objetivo de investigación O2 es proponer nuevas alternativas para la construcción de XML-DSLs.

Adicionalmente, en (Kosar, et al., 2016) se estableció que los lenguajes de propósito general más usados para la implementación de DSLs embebidos son Haskell (32,4%), Ruby (10,8%), Java y C++ (9,3), Scala (8,6%), Python (3,6%) y Clojure (2,9%); y existe un 20,9% que han utilizado herramientas varias. La creación de DSLs en un CSWE queda limitado al uso del lenguaje JavaScript y al uso de soluciones como Jison (Carter, 2009).

- **Enfoques de aplicación para DSL**

En (Kosar, et al., 2016) se determinó el enfoque de aplicación para DSL con los siguientes resultados: enfoque embebido 34,3%, compilador 28,1%, preprocesador 15,0%, intérpretes, COTS[7] 7,9%, híbridos 3,9% y compilador/intérprete extensible 2,9%. Los resultados no implican que la comunidad DSL se incline por el enfoque embebido o compilador, depende de gran manera de los objetivos que busca el DSL. En nuestro caso, el CSWE limita el enfoque a la creación de intérpretes.

2.5.2 MAPEO SISTEMÁTICO

Identificadas las tendencias de investigación, los posibles temas abiertos y la demografía de la literatura sobre los DSLs obtenida a partir del SMS de (Kosar, et al., 2016), en esta sección se realiza un Estudio del Mapeo Sistemático sobre XML-DSLs en clientes web a partir del 2008 para dar respuesta a las siguientes cuestiones:

RQ1. ¿Cuál es la producción en términos de publicaciones sobre los XML-DSLs en web?
RQ2. ¿Cuáles son los DSL Textuales, XML-DSLs y DSVLs previamente creados que hagan uso de especificaciones XML?
RQ3. ¿Cuáles son las herramientas usadas con mayor frecuencia para la creación de intérpretes de XML-DSL en cliente web?

Así, para dar respuesta a las cuestiones planteadas, se define la siguiente cadena de búsqueda primaria:

CB: ("domain-specific language" OR "DSL") AND ("XML" OR "language XML")
AND (year>2007) AND "web".

La CB fue aplicada en Agosto de 2016, sobre las siguientes bases de datos: IEEE Xplore Digital Library (ieeexplore.ieee.org), Science Direct (www.sciencedirect.com) y ACM Digital Library (dl.acm.org). En la Tabla 2-3 se muestra el total de resultados de los estudios (121 estudios) con la CB. El trabajo previo mencionado en de la sección 1.2.4 (Chavarriaga & Macías, 2009), se encuentra como parte del listado proporcionado por Science Direct.

La CB se adaptó a cada una de las fuentes y se aplicó la búsqueda sobre el resumen, filtrando conferencias, magazines y revistas. Se realizó la *lectura de resúmenes* e *introducción*, y para algunos casos se realizó una lectura completa para determinar la clasificación de las publicaciones. Para las

[7] Commercial Off-The-Shelf (COST). Herramientas y metodologías aplicadas a un dominio específico.

publicaciones candidatas, se determinó si el estudio primario definía o usaba DSLs, lenguajes XML y/o compiladores o intérpretes.

De las 118 publicaciones de la Tabla 2-3, en 70 estudios (59%) se definen o usan DSLs, intérpretes, o tratan simplemente sobre temas relacionados con XML y la web. El resto de publicaciones (48 estudios, 41%) tratan sobre metodologías, estudios de aplicaciones, algoritmos, tutoriales, optimización de XML, web semántica y aplicaciones móviles, entre otros. Estos estudios fueron clasificados en la columna “Otros”.

Tabla 2-3. Lista de resultados para el SMS con la cadena de búsqueda CB.

Bases de datos	Sigla	Publicaciones	Candidatas	Otros
IEEE Xplore Digital Library	IEEE	47	29	18
Science Direct	SD	30	21	9
ACM Digital Library	ACM	41	20	21
Total		**118**	**70**	**48**

Se trabajo con las 118 publicaciones iniciales para determinarlos siguientes aspectos: (1) el tipo de publicación (J: Revista, C: Conferencia, O: Otros); (2) el año de publicación; (3) el tipo de enfoque (T: Teórico, P: Práctico, E: Encuestas y otros); (4) el tipo de DSL (DSL Textual, XML-DSL y/o DSVL); (5) origen de la ejecución DSL (Escritorio, Servidor y/o Navegador) (6) IDEs y *frameworks* utilizados (Eclipse, Visual Studio, Otros y Desconocido); y (7) lenguajes de programación utilizados (Java, Scala, C#, JavaScript, Clojure, Jolie, Ruby, PHP, Otros y Desconocido).

En la Tabla 2-4 se muestra la lista completa de publicaciones candidatas (70 estudios). Se presenta en orden alfabético por autor y año, el tipo de publicación, el tipo de DSL, el origen de ejecución y los IDEs/*frameworks* utilizados.

Tabla 2-4. Lista de publicaciones Candidatas del SMS.

			Tipo de DSL			Ejecución DSL			Herramientas/IDEs			
Número	Artículo por autor	Tipo Public.	DSL Textual	XML-DSL	DSVL	Escritorio	Servidor	Navegador	Eclipse	Visual Studio	Otros	Desconocido
1	(Abiteboul, et al., 2009)	J		✓		✓						✓
2	(Abiteboul, et al., 2008)	C		✓		✓						✓
3	(Aghaee & Pautasso, 2014)	J	✓			✓			✓			
4	(Agreiter, et al., 2008)	J		✓			✓		✓			
5	(Ardito, et al., 2014)	J	✓	✓	✓	✓						✓
6	(Ardito, et al., 2014)	J			✓	✓					✓	
7	(Bashir, et al., 2013)	C		✓		✓						✓
8	(Bernardino, et al., 2016)	C	✓			✓					✓	
9	(Boyer, 2008)	C		✓			✓					✓
10	(Boyer, et al., 2008)	C		✓			✓					✓
11	(Cadavid, et al., 2009)	C	✓									✓
12	(Cao & Yang, 2011)	C		✓		✓	✓					✓
13	(Carvalho, et al., 2014)	C	✓					✓	✓			
14	(Chatley, et al., 2010)	C	✓			✓			✓			
15	(Chavarriaga, 2009)	O		✓			✓			✓		
16	(Chavarriaga & Díez, 2008)	C		✓			✓			✓		
17	(Chavarriaga & Macías, 2009)	C	✓		✓		✓			✓		
18	(Chavarriaga & Macías, 2008)	J	✓		✓		✓			✓		

Nota: **Tipo de publicaciones**: Revistas (J); Conferencias/Congresos (C); Otros estudios (O).

Tabla 2-4. (Continuación) Lista de publicaciones Candidatas del SMS.

			Tipo de DSL			Ejecución DSL			Herramientas/IDEs			
Número	Artículo por autor	Tipo Public.	DSL Textual	XML-DSL	DSVL	Escritorio	Servidor	Navegador	Eclipse	Visual Studio	Otros	Desconocido
19	(Chlipala, 2015)	C	✓			✓			✓			
20	(Chlipala, 2015)	C	✓			✓			✓			
21	(Conejero, et al., 2015)	J	✓			✓						✓
22	(Constantin, et al., 2013)	C		✓			✓					✓
23	(Delac, et al., 2012)	J		✓			✓					✓
24	(Demirkol, et al., 2012)	C	✓			✓			✓			
25	(Díaz, et al., 2008)	J	✓	✓			✓		✓			
26	(Djuric, et al., 2010)	C	✓				✓				✓	
27	(Dwijatmiko & Nguyen, 2014)	C	✓			✓						✓
28	(Englert & Lam, 2009)	C	✓				✓					✓
29	(Figueiredo, et al., 2016)	C	✓			✓					✓	
30	(Firmenich, et al., 2014)	J		✓				✓				✓
31	(Groenewegen & Visser, 2008)	J	✓				✓		✓			
32	(Groenewegen, et al., 2010)	J	✓				✓		✓			
33	(Groenewegen & Visser, 2013)	J	✓			✓			✓			
34	(Groenewegen & Visser, 2009)	C	✓				✓		✓			
35	(Havelund, 2014)	C	✓	✓		✓			✓			
36	(He, et al., 2013)	C			✓	✓					✓	
37	(He, et al., 2012)	C			✓	✓					✓	
38	(Hermida, et al., 2016)	J	✓			✓			✓			
39	(Herzberg, et al., 2009)	J		✓			✓		✓			
40	(Hlaoui & Benayed, 2008)	C			✓	✓						✓
41	(Hu & Yan, 2010)	C	✓				✓					✓
42	(Jacob, et al., 2014)	J	✓			✓			✓			
43	(Karam, et al., 2008)	J			✓	✓						✓
44	(Lachgar & Abdali, 2015)	C	✓	✓		✓			✓			
45	(Liu, et al., 2010)	C	✓	✓	✓		✓		✓			
46	(Liu, et al., 2009)	C	✓				✓					✓
47	(Liu, et al., 2010)	C	✓			✓			✓			
48	(Mahmoudi & Mourlin, 2014)	C		✓		✓						✓
49	(Mahmoud, et al., 2014)	C			✓		✓					✓
50	(Maximilien, et al., 2008)	J	✓				✓				✓	
51	(Maeda, 2009)	J	✓				✓					✓
52	(Montesi, et al., 2015)	J	✓				✓	✓	✓			
53	(Oliveira, et al., 2010)	C			✓	✓				✓		
54	(Oren, et al., 2008)	C		✓			✓		✓			
55	(Oyarzun, et al., 2009)	C		✓			✓					✓
56	(Prud, et al., 2014)	C	✓	✓		✓					✓	
57	(Sommaruga, et al., 2011)	C		✓		✓						✓
58	(Sarasa & Sierra, 2015)	C	✓				✓					✓
59	(Satu, et al., 2015)	J		✓		✓	✓					✓
60	(Shim, et al., 2009)	C		✓		✓	✓				✓	
61	(Shu, et al., 2008)	J		✓			✓					✓
62	(Taddesse & Chbeir, 2010)	C		✓			✓		✓			
63	(Tamayo, et al., 2011)	C		✓		✓			✓			

Nota: **Tipo de publicaciones**: Revistas (J); Conferencias/Congresos (C); Otros estudios (O).

Tabla 2-4. (Continuación) Lista de publicaciones Candidatas del SMS.

			Tipo de DSL			Ejecución DSL			Herramientas/IDEs			
Número	Artículo por autor	Tipo Public.	DSL Textual	XML-DSL	DSVL	Escritorio	Servidor	Navegador	Eclipse	Visual Studio	Otros	Desconocido
64	(Tarazona, et al., 2014)	J			✓	✓			✓			
65	(Thies & Vossen, 2009)	C		✓	✓		✓					✓
66	(Thomo & Venkatesh, 2008)	C	✓			✓						✓
67	(Thu Thuy, et al., 2009)	C	✓				✓					✓
68	(Tsalaportas, et al., 2015)	J		✓			✓					✓
69	(Vrieze, et al., 2011)	J			✓	✓						✓
70	(Wu & Li, 2012)	C		✓			✓	✓	✓			
71	(Yahya, et al., 2015)	C			✓	✓					✓	
72	(Zampoglou, et al., 2013)	C		✓		✓					✓	
73	(Zhang & Hölzl, 2015)	J	✓			✓			✓			
	Total		**35**	**31**	**13**	**36**	**32**	**4**	**26**	**2**	**11**	**31**

Nota: **Tipo de publicaciones**: Revistas (J); Conferencias/Congresos (C); Otros estudios (O).

En estos trabajos se estableció 35 DSLs textuales, 31 XML-DSLs y 13 DSVLs, distribuidos en su origen de ejecución en 36 de escritorio, 32 en servidores (web, base de datos), y en navegador o cliente web 4. Además, se estableció, que 26 fueron implementados con Eclipse, 2 con Visual Studio, 11 otros (IBM Share Code, GenX, GenJ, We2Animation, Betrock, entre otros) y 31 desconocidos.

En la Figura 2-3(a) se muestra el número de publicaciones por tipo de publicación (revista y conferencias/congresos). Es interesante resaltar que es similar la diferencia entre las publicaciones de revistas (65%) y las publicaciones de conferencias/congresos (65%), respecto al SMS de (Kosar, et al., 2016), las publicaciones en revistas del 34%, y 66% de conferencias/congresos.

En la Figura 2-3(b) se muestra el número de publicaciones por fuentes de distribución (según Tabla 2-3), en donde IEEE Library y ACM Digital Library acumulan el 75% de las publicaciones (88 de las 118 publicaciones).

En la Figura 2-3(c) se muestra el número de publicaciones por año. En los años 2008, 2009 y 2014 se aprecia el mayor número de publicaciones con el 47% de publicaciones (55 estudios).Por último, en la Figura 2-3(d) se muestra el número de publicaciones por año y por tipo de publicación. Se observa una tendencia a la baja en conferencias/congresos, en los primeros años, con una leve recuperación en el 2013 y 2014. Mientras que la media de publicaciones en revistas es de 4,6 publicaciones, y en el 2016 ya aparecen 4 publicaciones a la fecha del SMS (Agosto 2016).

En la Figura 2-4 se muestra la distribución de las 118 publicaciones candidatas (color azul, 70 estudios) y otros estudios (color rojo, 48 estudios). En las publicaciones candidatas definen o utilizan DSLs Textuales, XML-DSLs y DSVLs, o combinación entre ellas. Es de mencionar que en (Ardito, et al., 2014) y (Liu, et al., 2010) combinan los tres tipos de DSL. Mientras que en los otros estudios se enfocan en la creación de servicios web integrados, en la optimización de lectura y búsqueda en documentos XML, en el almacenamiento e intercambio de información XML y la creación tutoriales y usos de lenguajes XML (SVG, XML-FO, PDFX, MathML, XSLT, etc.).

De aquí en adelante, nos centraremos en discutir sobre resultados encontrados sobre la lista de publicaciones candidatas de la Tabla 2-4.

(a) DISTRIBUCIÓN POR TIPO DE PUBLICACIÓN

Revistas (41 pub.) 35%

Conferencias/Congresos (77 pub.) 65%

(b) NÚMERO DE PUBLICACIONES POR FUENTE

IEEE Xplore Digital Library (47 pub.) 40%

ACM Digital Library (41 pub.) 35%

Science Direct (30 pub.) 25%

(c) DISTRIBUCIÓN DE PUBLICACIONES POR AÑO

Año	2008	2009	2010	2011	2012	2013	2014	2015	2016
Publicaciones	20	18	10	10	9	13	17	13	8

Promedio (13)

(d) DISTRIBUCIÓN DEL TIPO DE PUBLICACIONES POR AÑO

Promedio (8,5)

Promedio (5,5)

Revistas — Conferencias/Congresos

Figura 2-3. Distribución por tipo de publicación (revista, conferencias/congresos y trabajos previos), por fuente de distribución y publicaciones por año que mencionan DSLs o temas relacionados.

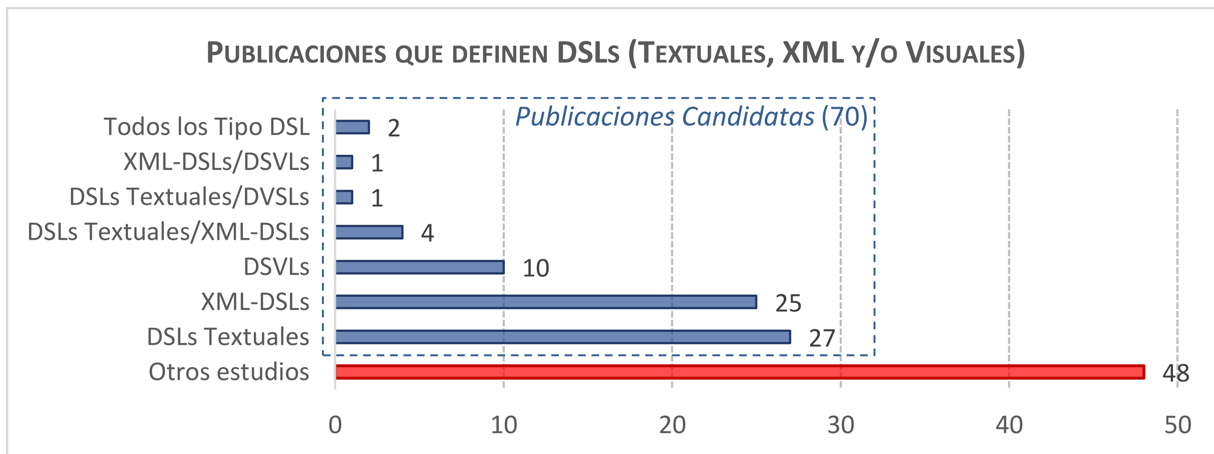


Figura 2-4. Número de publicaciones candidatas (color azul) y otros estudios (color rojos). Distribución de publicaciones candidatas por tipo de DSL.

■ **Publicaciones XML-DSLs**

Para las preguntas planteadas anteriormente (RQx) nos centraremos en las publicaciones que incluyen XML-DSLs (25 publicaciones), DSLs Textuales y XML-DSLs (4 publicaciones). Es decir, un total de 29 publicaciones, que denominaremos **Publicaciones XML-DSLs**. Excluimos los XML-DSLs que están relacionadas con lenguajes visuales y DSVLs, están serán analizadas posteriormente.

En la Figura 2-5(a) se muestra la distribución de las Publicaciones XML-DSLs para el periodo 2008-2016 por tipo de publicación, año y fuente de información. Se observa la ausencia de trabajos en revistas para los años 2010, 2011, 2013 y 2016, y para el resto de años del período 2008-2016 no alcanza a un promedio de 2 publicaciones. En cambio en conferencias/congresos se mantiene un promedio de 2,5 publicaciones en el período 2008-2016 (21 publicaciones, 72%). En la Figura 2-5(b) se muestra la distribución de los estudios en las diferentes fuentes, y en la Figura 2-5(c) se muestra que en 4 estudios se definieron tanto DSLs textuales como XML-DSLs (Díaz, et al., 2008; Havelund, 2014; Lachgar & Abdali, 2015; Prud, et al., 2014).

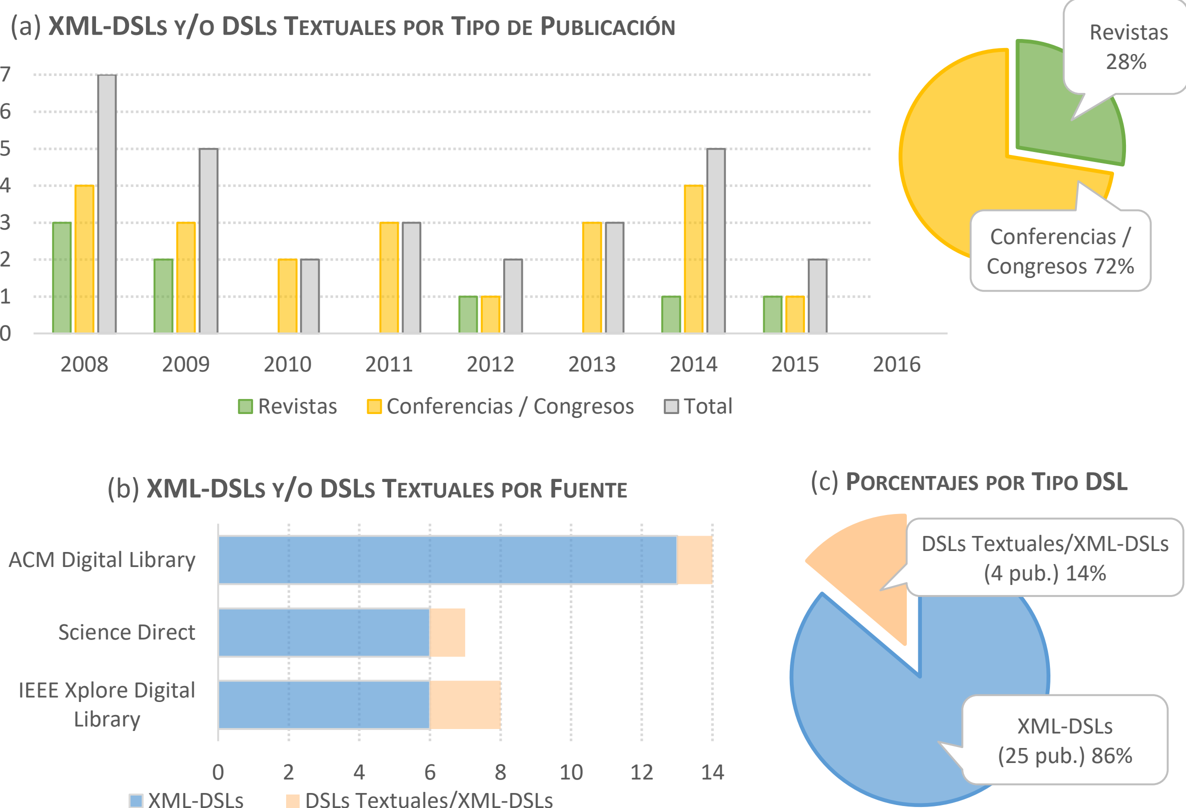


Figura 2-5. XML-DSLs y/o DSLs Textuales por tipo de publicación (revista, conferencias/congresos y trabajos previos) y por fuente de distribución para las Publicaciones XML-DSLs.

En la Figura 2-6(a) y (b) se muestra la distribución de IDEs para las Publicaciones XML-DSLs. El 55% (5 publicaciones) no se puede determinar el IDE utilizado, mientras que el 7% (2 publicaciones) usan herramientas como XMLSky, We2Animation y GenX.

Es de destacar que Visual Studio no es utilizado para la creación de XML-DSLs, solamente es usado en las 3 publicaciones de los estudios previos realizados en la sección 1.2.4. En 9 publicaciones (33%) se infiere el uso de Eclipse.

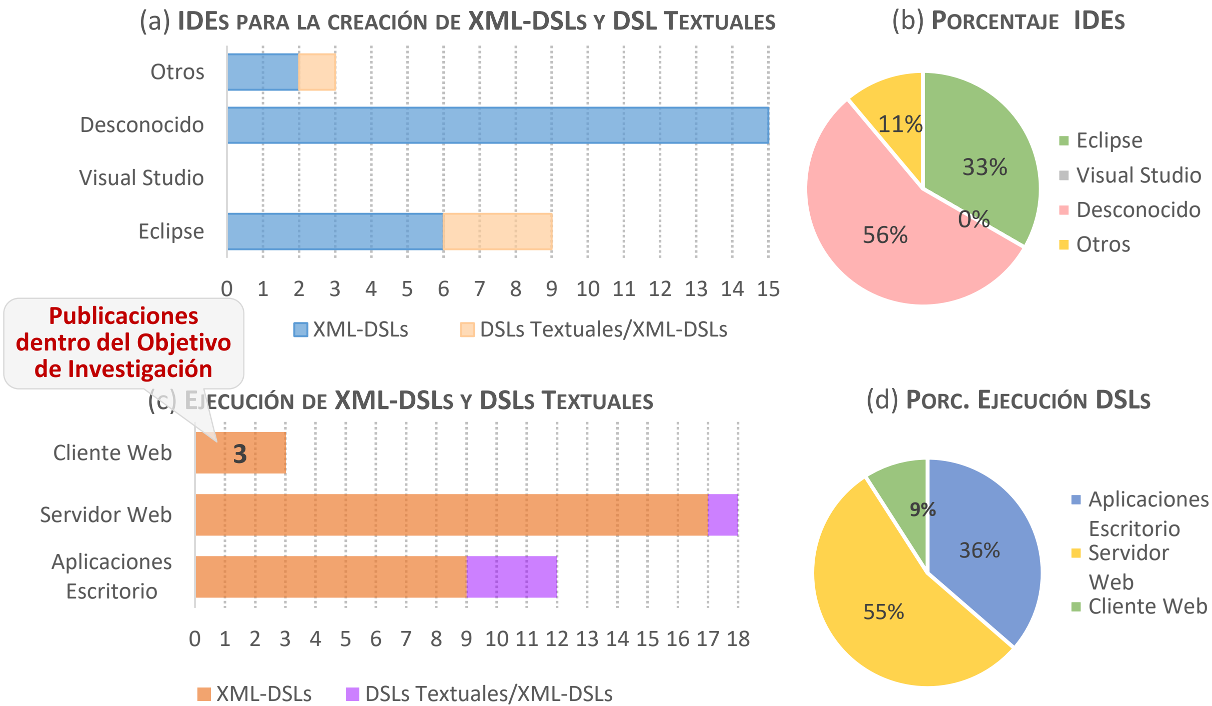


Figura 2-6. IDEs para la creación de DSLs y formas de ejecución de las Publicaciones XML-DSLs.

En la Figura 2-6(c) y (d) se muestra el origen de ejecución o evaluación de los XML-DSLs y los DSL Textuales de las Publicaciones XML-DSLs. El 9% restante (3 XML-DSLs en 3 publicaciones) son evaluadas sobre un cliente web y en particular son las que cumplen con los objetivos de las preguntas de investigación RQx para nuestro SMS. A continuación damos un resumen de estas publicaciones:

1. Herzberg, D.; Marsden, N.; Kübler, P.; Leonhardt, C.; Thomanek, S. Jung, H.; Becker, A. *Specifying computer-based counseling systems in health care: A new approach to user-interface and interaction design*. Journal of Biomedical Informatics, 42(2), 2009, p. 347-355. El objetivo del artículo es facilitar el diseño estructurado para la implementación de sistemas de asesoramiento para el cuidado de la salud humana. Involucra la definición de un XML-DSLs para describir y definir las interfaces de usuario e interactuar entre los diferentes usuarios (médicos, pacientes, administrativos, etc.).
2. Firmenich, S.; Rossi, G.; Winckler, M.; Palanque, P. *An approach for supporting distributed user interface orchestration*. International Journal of Human-Computer Studies, 72 (1), 2014, p. 53-76. El objetivo de este artículo es crear interfaces de usuario distribuidos que ofrezcan procedimientos para orquestar las tareas del usuario a través de múltiples sitios web. Se crea el lenguaje DUI y es un XML-DSL para describir interfaces, composición de tareas y aumentadores web.
3. Shim, H.; Kang, B.; Kwag, K. *Web2Animation - Automatic Generation of 3D Animation from the Web Text*. In Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology – vol. 01 (WI-IAT '09), Vol. 1. IEEE Computer Society, Washington, USA, 596-601. El objetivo del trabajo es presentar un sistema de producción de animación 3D basado en una historia en lenguaje natural, los textos relacionados son analizados para construir un conjunto de comandos de animación y crear una acción. Se usa una declaración XML para para definir las diferentes representaciones de animación sobre la web.

En términos generales, las tres publicaciones definen XML-DSLs sobre un cliente web. Se concluye, aunque no está de forma explícita, que es una implementación ad-doc y no se utilizó ningún tipo de herramienta para su creación. Adicionalmente, el SMS *no encontró intérpretes sobre CSWE para la evaluación de XML-DSLs*.

- **Publicaciones DSVLs**

Como está construida la CB del SMS, permite establecer algunas publicaciones en las cuales se han hecho trabajados relacionados con lenguajes visuales y DSVLs. De las 70 publicaciones candidatas de la Tabla 2-4, el 18% (13 publicaciones) hacen referencia a lenguajes visuales y DSVLs, y se denominarán **Publicaciones DSVLs**.

En la Figura 2-7(a) se muestra la distribución de las Publicaciones DSVLs en el periodo 2008-2016 por tipo de publicación, año y fuente de información. El 43% (6 publicaciones) se publican en revistas y el 57% (9 publicaciones) son publicadas para conferencias y/o congresos. En la Figura 2-7(b) se muestra la distribución de los DSVLs versus fuentes de distribución, y en la Figura 2-7(c) por tipo de DSL.

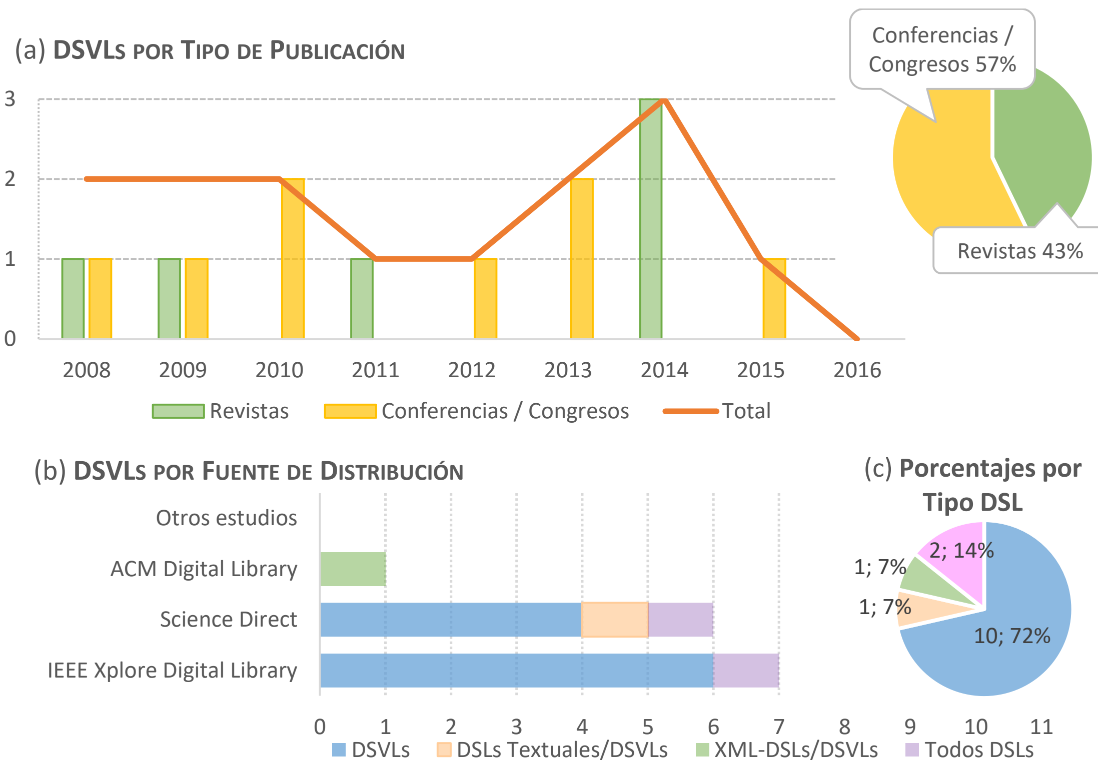


Figura 2-7. DSVLs y/o DSLs Textuales y/o XML-DSLs por tipo de publicación (revista, conferencias/congresos y trabajos previos) y por fuente de distribución para las Publicaciones DSVLs.

En la Figura 2-8(a) y (b) se muestra la distribución de IDEs para las Publicaciones DSVLs. El 50% (7 publicaciones) no se puede determinar el IDE utilizado, mientras que el otro 50% (7 publicación) usa usan algún tipo de herramienta/IDE. Nuevamente, es de destacar que Visual Studio no es utilizado para la creación de DSVLs, solamente es usado en las 2 publicaciones de los estudios previos realizados.

En la Figura 2-8(c) y (d) se muestra el origen de ejecución o evaluación de los diferentes tipos de DSLs para las Publicaciones DSVLs. Todos los DSVLs son definidos para herramientas de escritorio pero su ejecución el 45% se hace sobre un servidor web, mientras el 55% se hace sobre aplicaciones de escritorio. El objetivo en este análisis es determinar si existen DSVLs que se ejecuten en clientes web, pero no se encontró algún indicio. Tampoco se encontró indicios de creación de DSVLs en clientes web.

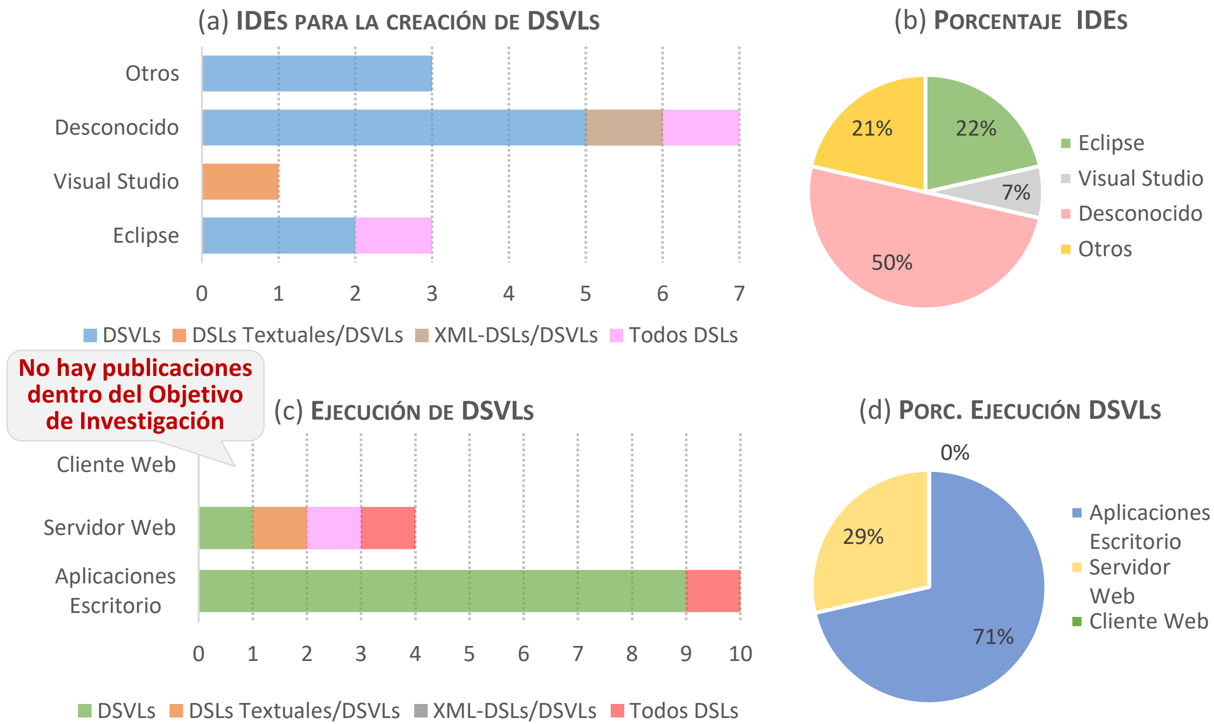


Figura 2-8. IDEs para la creación de DSVLs y formas de ejecución de las Publicaciones DSVLs.

2.6 Resumen

En la Figura 2-9 se muestra el resumen del Estado del Arte. La primera parte se dedicó resumir los *conceptos básicos* relacionados con nuestra investigación. A continuación, se profundizó sobre *los lenguajes* XML, los DSLs y lenguajes visuales. Tras esto, se revisaron los *frameworks e IDEs* de los principales entornos de desarrollo integrado y entornos de modelado visual para la creación de DSLs y DSLs textuales, y los *frameworks* para la creación de diagramas y lenguajes visuales en CSWE. Por último, tras determinar las funcionalidades deseadas para la creación de diagramas programables y lenguajes visuales, se llevó a cabo un mapeo sistemático sobre DSLs textuales, XML-DSLs y DSVLs en CSWE.

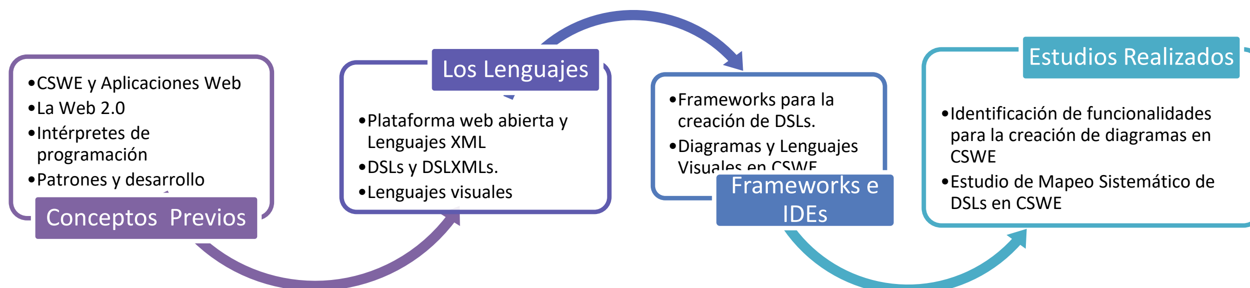


Figura 2-9. Resumen del Estado del Arte.

En el estudio de creación de diagramas programables y lenguajes visuales se hace un análisis comparativo de herramientas para la gestión de modelos de las *frameworks* para escritorio (GMF/Graphit, VSSDK y VLDesk) y para aplicación web (gliffy, LucidChar, yFiles, GoJS, graphviz y cacoo). En el estudio sobre los elementos necesarios para la creación de diagramas programables y lenguajes

visuales se hace un análisis comparativo de herramientas para la gestión de modelos de los frameworks para escritorio (GMF/Graphit, VSSDK y VLDesk) y para aplicación web (gliffy, LucidChar, yFiles, GoJS, graphviz y cacoo). En términos generales, las herramientas de gestión de diagramas y de interacción están muy elaboradas e integran interfaces de usuario agradables y usables. Sin embargo, no todas aportan renderización de diagramas, modelos gráficos y lenguajes visuales. Asimismo, parece que la asociación de fuentes de información heterogénea (XML/JSON) no es primordial para estas herramientas. Además, las herramientas para la construcción de diagramas como GMF/Graphit, VSSDK y VLDesk requieren de implementación ad-hoc para la inclusión de elementos visuales. Por último, en las herramientas analizadas no se contempla la funcionalidad de recomendación de diagramas, que constituye un requisito interesante a incluir en la creación de modelos gráficos.

El Estudio de Mapeo Sistemático (SMS, Sistematic Mapping Study) de (Kosar, et al., 2016) enmarca nuestro trabajo en áreas poco exploradas, como la creación de nuevos intérpretes y la creación de herramientas web para la implementación y evaluación de XML-DSLs en un CSWE. El uso de lenguajes como JavaScript alimenta las expectativas para crear soluciones de software interesantes sobre clientes web. El intérprete para evaluar XML-DSLs en un CSWE (O1) es un área poco estudiada e innovadora.

El Estudio del Mapeo Sistemático sobre XML-DSLs llevado a cabo en la sección XX, hace una revisión de los XML-DSLs sobre CSWE. Este estudio ha permitido identificar que efectivamente existen pocas soluciones para la creación y evaluación de XML-DSL en CSWE. Las pocas aproximaciones existentes son implementaciones ad-hoc. Además, no se han encontrado indicios de intérpretes y/o herramientas para la creación XML-DSLs sobre CSWE. Es cierto que los IDEs actuales ofrecen herramientas potentes para la creación de XML-DSLs, pero el objetivo de investigación O2 es proponer nuevas alternativas para la construcción de XML-DSLs sobre CSWE.

Adicionalmente, el Mapeo Sistemático realizado ha permitido identificar el número reducido de publicaciones relacionadas con la creación y definición de lenguajes visuales, cuya especificación e implementación quedan limitadas a aplicaciones de escritorio. Esto sustenta la necesidad del objetivo O3 para definir e implementar DSVLs, diagramas y generación automática de diagramas en un CSWE.

Capítulo 3. PsiEngine: Motor de Evaluación Psi

Este capítulo define y desarrolla el motor de evaluación de programas escritos en DSL con gramática XML (denominado **Motor de Evaluación Psi**) en un CSWE (O1). Se presentará una visión general del Motor y algunos elementos de programación necesarios para su implementación. A continuación se formalizarán las gramáticas Psi, los lenguajes Psi y los componentes Psi. Además, se dedicará una sección a formalizar e implementar el intérprete del Motor. Por último, se presentará una metodología para el diseño e implementación de estos lenguajes y componentes, acompañado de un caso de estudio (lenguaje MiniPsi).

3.1 Introducción

En un CSWE, los lenguajes XML como XSL, SVG, MathML, etc., tienen como objetivo enriquecer el contenido de una página web y suelen estar incorporados de forma nativa en los navegadores actuales, es decir, saben cómo manipula el DOM de esos lenguajes XML. En términos generales, cada elemento (etiqueta) de un lenguaje XML, tiene una funcionalidad asociada para alcanzar su objetivo (por ejemplo, crear un círculo, rectángulo, línea, etc., con lenguaje SVG). La ejecución de todas las funcionalidades en un programa escrito con el lenguaje XML es la solución a un problema (por ejemplo, crear diagramas o figuras con SVG). Una característica importante de los lenguajes XML en un CSWE, es la capacidad de adicionar, eliminar y modificar fragmentos de código XML de forma dinámica mediante el DOM respectivo empleando el lenguaje JavaScript, permitiendo actualizar de forma inmediata el contenido de la página web.

En la Figura 3-1 se muestra el diagrama de bloques general del Motor de Interpretación y Evaluación de Soluciones Programables o **Motor de Evaluación Psi** (**PsiEngine**, **P**rogrammable **S**olutions **I**nterpreter **Engine**) en un CSWE. El PsiEngine puede definirse del siguiente modo:

> **PsiEngine**
>
> Motor de evaluación que combina los lenguajes HTML5, CSS3, JavaScript y DOM, con las tecnologías, servicios y herramientas de la Web 2.0, y con la definición de XML-DSL para la construcción de componentes web, widgets web y/o páginas web dinámicas que permitan dar solución a problemas específicos presentes en una aplicación web o en partes de ella.

En términos generales, el PsiEngine facilita crear nuevos XML-DSLs, asociar su funcionalidad y proporcionar un entorno de trabajo y de evaluación.

El PsiEngine visto como caja negra (ver Figura 3-1) tiene, como entradas, aquellos programas escritos en uno o más XML-DSLs (denominados **Lenguajes Psi**) y diferentes tipos de *Recursos*. Su resultado es obtener una funcionalidad determinada de una aplicación web permita dar solución a un problema específico. Un **Lenguaje Psi** $\mathbb{L}_j$ es un XML-DSL que tiene asociada una gramática XML (árbol de etiquetas y definición de atributos) y una funcionalidad para cada etiqueta, con capacidad de asociar información heterogénea (XML/JSON) de forma nativa y de ejecutar código *inline* JavaScript.

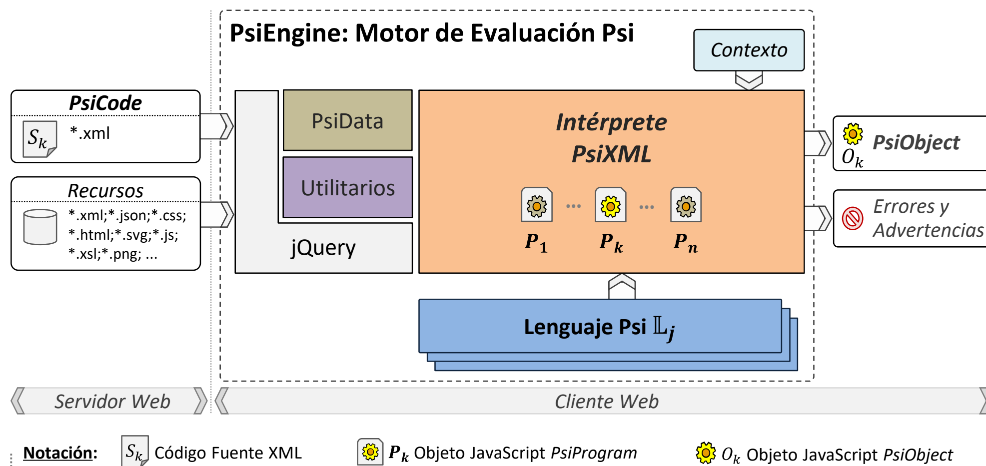


Figura 3-1. Motor de Interpretación y Evaluación de Soluciones Programables o Motor de Evaluación Psi (PsiEngine, Programmable Solutions Interpreter Engine).

Para entrar algo más en los detalles de ejecución, tanto los *Recursos* (por ejemplo, ficheros CSS3, XSL, XML, frameworks JavaScript, widgets, diagramas en SVG, fórmulas en MathML, fragmentos de páginas, etc.), como el *Contexto* (información disponible para su evaluación, usualmente de configuración) y los códigos fuentes escritos en Lenguaje Psi (*PsiCode*, código fuente Psi), son interpretados y evaluados por la pieza central del PsiEngine, el **Intérprete PsiXML** (denotado **PsiXML**). El resultado es una funcionalidad encapsulada en un objeto JavaScript (*PsiObject*, código objeto Psi). El proceso de interpretación y evaluación, se denomina ejecución de un programa Psi P_k, consiste en cargar el código fuente Psi obtener el árbol de etiquetas, recorrer y evaluar cada funcionalidad asociada, y al final obtener un objeto *PsiObject*.

PsiXML es un *framework* JavaScript liviano y genérico, que interpreta y evalúa el código fuente Psi S_k en cualquier navegador de cualquier dispositivo. Para la implementación de PsiXML se utilizó jQuery (jquery.com) para facilitar la navegación por el DOM de un programa escrito en lenguaje Psi y ejecutar cada funcionalidad asociada. Es decir, el PsiXML trabaja con la misma filosofía de un lenguaje XML en un CSWE, aprovecha el DOM del *PsiObject* O_k, para modificar dinámicamente el *PsiCode* S_k original, e incluye poder adicionar, eliminar o modificar fragmentos de código y evaluarlos. Además, puede obtener *PsiCode* S_k modificado para su posterior uso, es decir, realizar el proceso de serialización de un nuevo programa Psi S_k.

A diferencia de los lenguajes XML en un CSWE, PsiXML tiene la capacidad de *asociar información* XML/JSON externa a elementos de un Psi Language $\mathbb{L}_j$ para usarla y modificarla en tiempo de ejecución en un área común denominada **PsiData**. En el CSWE, asociar información XML implica que existe el DOM de un XML, y que un elemento del *Psi Code* S_k tenga una referencia a elementos de ese DOM. De forma similar, es posible asociar información en JSON para elementos del *Psi Code* S_k. Es decir, se está

combinando la capacidad y rapidez de tener información JSON con la productividad y facilidad de los lenguajes XML.

Una de de las razones de peso para escoger XML-DSLs en un CSWE, ha sido la facilidad de manipular el DOM con JavaScript, y en particular trabajar con *frameworks* que facilitan este trabajo. Además, permite combinar lenguajes y tener disponibles herramientas como XSL para crear soluciones dinámicas, por ejemplo, crear *PsiCode* S_k a partir de transformaciones de información XML. Es decir, se agrega tanto contenido como funcionalidad para solucionar problemas de dominio específico.

3.2 CONCEPTOS DE PROGRAMACIÓN JAVASCRIPT

Está sección tiene como objetivo recopilar los conceptos de programación JavaScript que se utilizarán en la definición del lenguaje Psi, del componente Psi, del PsiXML y en la definición del modelo de programación y generación de código del Capítulo 4.

3.2.1 ELEMENTOS BÁSICOS DE PROGRAMACIÓN EN JAVASCRIPT

En lenguajes bajo la especificación ECMAScript (ECMA, 2011) y en particular JavaScript, una **variable** (var) puede verse como un contenedor de información. El **tipo de dato** está asociado al conjunto de valores que puede tomar y no a la variable. Por ejemplo, una variable x en un momento dado puede estar definida como un número (Number) y más adelante definida como una cadena (String), una fecha (Date), un objeto o inclusive una función.

Una **función** (function) es un bloque de instrucciones que admite argumentos y retorna el valor de una variable. Dentro de un bloque de instrucciones se pueden definir variables, hacer uso de operadores y estructuras de control que determinan el comportamiento de la función y su resultado. En JavaScript las funciones son objetos en sí mismos, es decir, se pueden definir propiedades y métodos, y son la base para construir los conceptos de clase y componente.

Por otro lado, un **objeto** (Object) en JavaScript es una *colección asociativa* con la inclusión de prototipo. Es decir, es un tipo de dato abstracto formado por una colección de claves únicas con sus respectivos valores. Las claves únicas son de tipo String, mientras que cada valor es el mismo tipo definido para una variable. Los elementos de la colección pueden ser creados, modificados o eliminados en tiempo de ejecución. Un **tipo enumerativo** se puede modelar con un objeto y sirve como un tipo de dato personalizado para que el código creado sea legible, sus valores son números o cadenas de caracteres. Una **tipo abstracto de datos** es información estructurada por campos, se puede modelar con un objeto que contiene exclusivamente información (números, cadenas de caracteres, booleanos, fechas, matrices, etc.), usualmente constituida por objetos anidados. Por último, una ***Matriz*** de valores (Array) es una colección de variables con un índice único numérico.

3.2.2 IMPLEMENTACIÓN DEL CONCEPTO CLASE

En JavaScript se usa la programación basada en prototipos y el concepto de **clase** no está presente. Estas se pueden obtener a través de la clonación de funciones ya existentes que sirven de prototipos, y su herencia se puede adquirir mediante la extensión de su funcionalidad (Crockford, 2008; White, 2009). En el FRAGMENTO 3.1 se ilustra este concepto suponiendo que se tiene cargada la librería de (Crockford, 2006).

En el ejemplo del FRAGMENTO 3.1 se muestra una clase base *Figure* con un método abstracto *area*. El objetivo del ejemplo es definir dos clases que hereden de *Figure* (las clases *Circle* y *Square*) y sobrecargar el método *area* para obtener el área del círculo y el cuadrado. De esta forma se pueden sumar áreas de diferentes tipos de figuras que estén definidas en una matriz.

FRAGMENTO 3.1. Ejemplo de creación de clases JavaScript con el uso de implementación de Crockford.

```
function Figure() { }; // Clase Base
Figure.methods({
  area: function () { return 0; } //Método abstracto
});

function Circle(r) { //Clase Heredada
  this.r = r; //Propiedad radio del círculo
}
Circle.inherits(Figure); // Implementación de herencia
Figure.methods({
  area: function () { return Math.PI * Math.pow(this.r, 2); } //Redefinición de método
});

function Square(l) { //Clase Heredada
  this.l = l; //Propiedad lado de cuadrado
}
Circle.inherits(Figure); // Implementación de herencia
Square.methods({
  area: function () { return Math.pow(this.l, 2); } //Redefinición de método
});

var figs = [new Circle(2), new Square(1), new Circle(1)],
    sum = 0;
for (var i = 0; i < figs.length; i++)
  sum += figs[i].area();
alert(sum);
```

En esta tesis, el FRAGMENTO 3.2 propone una función que facilita la creación dinámica de una clase (función *createClass*) en tiempo de ejecución para el CSWE.

FRAGMENTO 3.2. Función *createClass* para la creación dinámica de clases JavaScript.

```
function createClass(nameClass, constructor, properties, methods, inherits) {
  var define = nameClass + " = function(" + (constructor ? constructor : "") + "){ \n";

  //Adicionar propiedades
  for (var name in properties)
    define += "this." + name + " = " + properties[name] + ";\n";
  define += "\n};";

  //Adicionar herencia
  if (inherits)
    define += nameClass + ".inherits(" + inherits + ");\n";

  //Adicionar métodos
  if (methods)
    define += nameClass + ".methods( methods );\n";

  //Crear definición de clase
  eval(define);
}
```

Como ejemplo ilustrativo del uso de la función *createClass*, el FRAGMENTO 3.3 crea las mismas clases *Figure*, *Circle* y *Square*, obteniendo los mismos resultados.

FRAGMENTO 3.3. Ejemplo para la creación de clases con la función *createClass*.

```
createClass("Figure", null, null, { area: function () { } }, null);
createClass("Circle", "r", { r: "r" },
  { area: function () { return Math.PI * Math.pow(this.r, 2); } }, "Figure");
createClass("Square", "l", { l: "l" },
  { area: function () { return Math.pow(this.l, 2); } }, "Figure");
```

3.2.3 Implementación del concepto de Componente

Como se resumió en la sección 2.1.4, un **componente** reutilizable de programación debe ser identificable, accesible solo a través de su interfaz, tener servicios predefinidos, estar autodocumentado, y ser capaz de comunicarse con otros componentes. En JavaScript se encapsulará como una función estática sobre la que se pueden implementar los elementos de programación básicos, las clases y los subcomponentes (componente anidado para modularizar la funcionalidad del componente padre), y solo se publicarán aquellos elementos que se requieran. En el Fragmento 3.4 se muestra la plantilla JavaScript para definir un componente. Nótese que la interfaz pública del componente es un objeto.

Fragmento 3.4. Plantilla JavaScript para definir un componente como función estática.

```
var MyComponent = (function () {
  //Propiedades del componente
  //Métodos del componente
  //Elementos de programación
  return {
    //Interfaz de servicio: elementos públicos del componente
  };
})();
```

Las propiedades y métodos de un componente son variables globales en el ámbito del componente, es decir, son accedidas por cualquier elemento de programación definido dentro de él. En esta definición, la diferencia entre un método del componente y una función es puramente conceptual, ya que se definen de la misma manera. En el Fragmento 3.5 se muestra un ejemplo ilustrativo de un componente en JavaScript.

Fragmento 3.5. Ejemplo ilustrativo de un componente en JavaScript.

```
var MyComponent = (function () {
  var x, _y;  //Propiedades del componente
  function M1() { return "Hola Mundo!!";  };  //Métodos público del componente
  function _M2() { }; //Método privado del componente
  var O = {}; // Objeto del componente
  function C() { };
  C.methods({}); //Clase pública del componente
  //Otras definiciones de
  return {
    x: x, //Propiedad pública
    m1: M1, // Método público
    C: C // Clase pública
  };
})();

//Uso
var c = new MyComponent.C();
alert( MyComponent.m1() );
```

3.3 El Lenguaje Psi

Una vez establecidos los elementos de programación en JavaScript, esta sección está dedicada a formalizar los conceptos de **Lenguaje Psi** $\mathbb{L}$, **Gramática Psi** $\mathbb{G}$ y **Componente Psi** $\mathbb{K}$ dentro del PsiEngine (destacado en la Figura 3-2).

En términos generales, un **Lenguaje Psi** $\mathbb{L}$ es un XML-DSL que tiene asociada una estructura o árbol de etiquetas (Gramática Psi $\mathbb{G}$) y una funcionalidad para cada etiqueta (conjunto de clases asociadas del Componente Psi $\mathbb{K}$). En primera instancia se formalizará la Gramática Psi y a continuación se formalizarán el Lenguaje Psi y el Componente Psi.

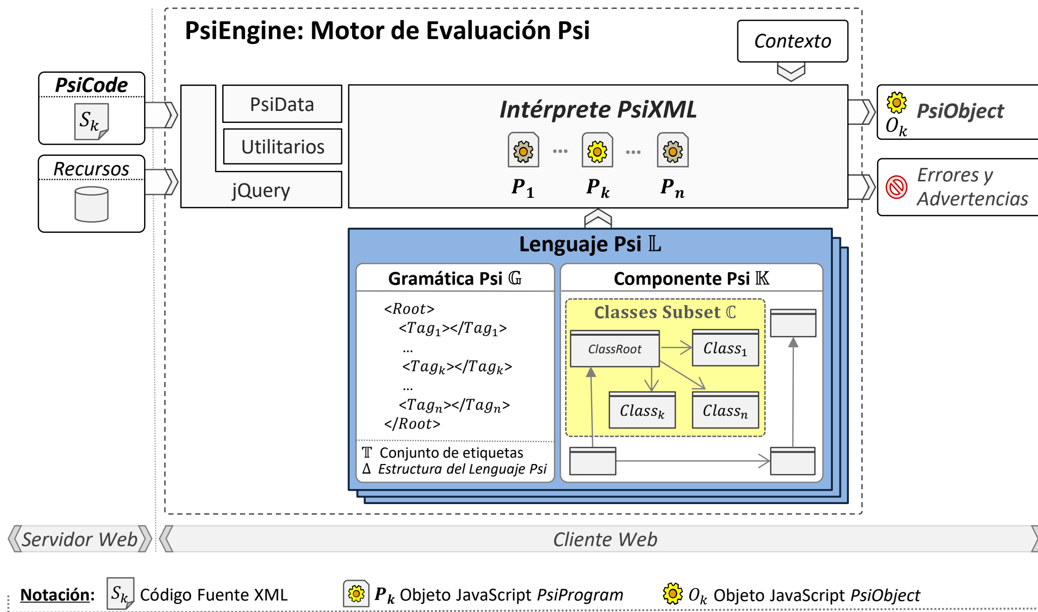


Figura 3-2. PsiEngine: Definición de Lenguaje Psi, Gramática Psi y Componente Psi

3.3.1 FORMALIZACIÓN DE LA GRAMÁTICA PSI

En general, una **gramática Psi** $\mathbb{G}$ (ver Figura 3-2) se denota con la tupla:

$$\mathbb{G} = \langle \mathbb{T} | Root | \Delta \rangle, \quad (3\text{-}1)$$

donde $\mathbb{T} = \{Root, Tag_1, Tag_2, \ldots, Tag_m\}$ es el conjunto de etiquetas, $Root$ es la etiqueta raíz y Δ es el objeto de la estructura gramatical definido como:

$$\Delta = \{Tag_k : \Delta_k | Tag_k \in \mathbb{T}\}, \quad (3\text{-}2)$$

donde $\Delta_k \in \Delta$ es un objeto especificado por:

$$\Delta_k = \{\text{TAG}: v_T, \text{CHILDREN}: v_H, \text{MULTIPLICITY}: v_M, \text{STRICT}: v_S, \text{VALIDATOR}: v_V\} \quad (3\text{-}3)$$

donde v_T es el nombre de la etiqueta, v_H es un objeto que define la lista de etiquetas hijas (por defecto $v_H = null$), v_M es la multiplicidad de las etiqueta con respecto a su etiqueta padre (posibles valores: "$0..1$", "$1..1$", "$0..n$" o "$1..n$", por defecto es $v_M =$ "$0..n$"), v_S define la validación estricta de las etiquetas hijas (es decir, no admite otro tipo de etiquetas, por defecto $v_S = false$), y v_V especifica la validación de atributos (por defecto $v_V = null$).

En la Figura 3-3(a) se define el *diagrama de la estructura de un lenguaje Psi* (Psi Language Structure Diagram), denotado por **PsiLSD**. El PsiLSD es la representación gráfica de Δ donde **<GrammarName>** es el nombre o sigla de la gramática $\mathbb{G}$, el cual se conecta con la etiqueta raíz $Root$ y su clase asociada de nombre $ClassRoot$. Además, para cualquier etiqueta Tag_k, se tiene que: "$x..y$" es la multiplicidad v_M de la etiqueta; $Class_k$ es nombre de la clase asociada; Tag_m hasta Tag_n es la lista de etiquetas hijas v_H con sus respectivas clases asociadas; y el círculo rojo significa validación estricta v_S de sus etiquetas (círculo blanco, caso contrario).

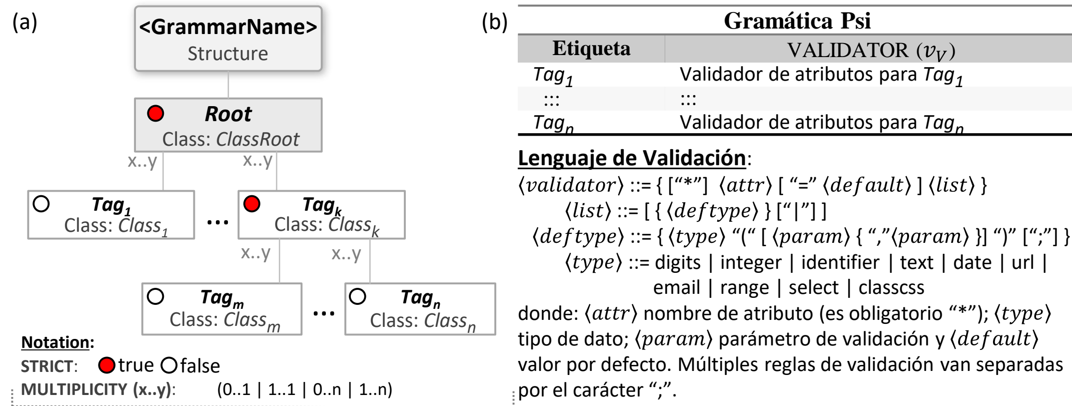


Figura 3-3. (a) Definición del diagrama de estructura del lenguaje Psi, denotado por PsiLSD. (b) El validador de atributos de la gramática Psi, denotada por PsiGVA.

En la Figura 3-3(b) se muestra el *validador de atributos de la gramática Psi* (Psi Grammar Validator Attributes), denotada por **PsiGVA**. El PsiGVA especifica el lenguaje de validación de atributos v_V para una etiqueta Tag_k. Véase el siguiente ejemplo ilustrativo:

$$v_V = \text{“} * amount = 200\text{: range}(0, 1000) \mid money\text{: select(Euro, Peso, Dolar)”}.$$

Significa que el atributo *amount* es un número obligatorio entre 0 y 1000, por defecto es 200, y *money* puede tomar los valores de “Euro”, “Peso” y “Dólar”.

Tanto PsiLSD como PsiGVA simplifican la especificación y el diseño de gramáticas Psi para el PsiEngine. En el FRAGMENTO 3.6 se muestra la plantilla JavaScript para definir una gramática Psi $\mathbb{G}$. En primer lugar se definen los identificadores del conjunto de etiquetas $\mathbb{T}$ (ID_TAG_*), a continuación se construye la estructura del lenguaje definiendo $\Delta_k \in \Delta$, para cada $Tag_k \in \mathbb{T}$. Por último se define el objeto GRAMMAR de la gramática Psi.

FRAGMENTO 3.6. Plantilla JavaScript para definir una gramática Psi.

```
// Conjunto de identificadores de la gramática Psi
var ID_TAG_T1 = "Root", ID_TAG_T1 = "T1", …, ID_TAG_Tm = "Tm";

// Estructura del lenguaje
var ID_TAGS = {};
ID_TAGS[ID_TAG_ROOT]={TAG:"Root", CHILDREN: …, MULTIPLICITY: …, STRICT: …, VALIDATOR: "…"};
ID_TAGS[ID_TAG_T1]={TAG:"Tag1", CHILDREN: …, MULTIPLICITY: …, STRICT: …, VALIDATOR: "…"};
…
ID_TAGS[ID_TAG_Tm]={TAG:"Tagm", CHILDREN: …, MULTIPLICITY: …, STRICT: …, VALIDATOR: "…"};

//definición de la gramática Psi
var GRAMMAR = {
  NAME: "GrammarName", // Nombre de la grámatica Psi
  TAGS: ID_TAGS, // Estructura del lenguaje Psi
  ROOT: ID_TAG_ROOT // Etiqueta raíz. Para algún identificador ID_TAG_Tk
};
```

3.3.2 FORMALIZACIÓN DEL LENGUAJE PSI Y EL COMPONENTE PSI

En un CSWE, un **Lenguaje Psi** define una gramática Psi a la que se le asocia un conjunto de clases dentro de un componente reutilizable. Esta asociación produce una funcionalidad que sirve como solución dentro de una aplicación web a un problema específico. Por lo tanto, un **lenguaje Psi** $\mathbb{L}$ se define como la tupla:

$$\mathbb{L} = \langle \mathbb{G} | \mathbb{K} | \mathbb{T} \leftrightarrow \mathbb{C} \rangle, \quad (3\text{-}4)$$

donde $\mathbb{G}$ es la gramática Psi definida en (3-1), $\mathbb{K}$ es un componente de software JavaScript reutilizable que se denominará **componente Psi**, $\mathbb{T}$ es el conjunto de etiquetas de la gramática Psi, $\mathbb{C} = \{ClassRoot, Class_1, \dots, Class_n\}$ es un subconjunto de clases implementadas en $\mathbb{K}$ que se denominará **clases asociadas** del lenguaje, y finalmente, $\mathbb{T} \leftrightarrow \mathbb{C}$ es la asociación entre una etiqueta Tag_k y una clase $Class_k$, para cada $Tag_k \in \mathbb{T}$ y $Class_k \in \mathbb{C}$, respectivamente.

En la Figura 3-4 se muestra el lenguaje Psi $\mathbb{L}$ y la asociación entre las etiquetas de la gramática $\mathbb{G}$ y el subconjunto de clases $\mathbb{C}$ dentro de un CSWE. Cabe reseñar que una clase de $\mathbb{C}$ puede estar asociada a más de una etiqueta de $\mathbb{T}$, esto significa que la funcionalidad es la misma para estas etiquetas. Del mismo modo, se puede tener el nombre de etiqueta en diferentes partes de la estructura gramatical pero su funcionalidad puede ser diferente, esto implica que debe tener diferentes clases asociadas. Por otro lado, en la Figura 3-3(a) se puede observar que el PsiLSD también contempla la asociación del nombre de la clase $Class_k$ a su correspondiente etiqueta Tag_k.

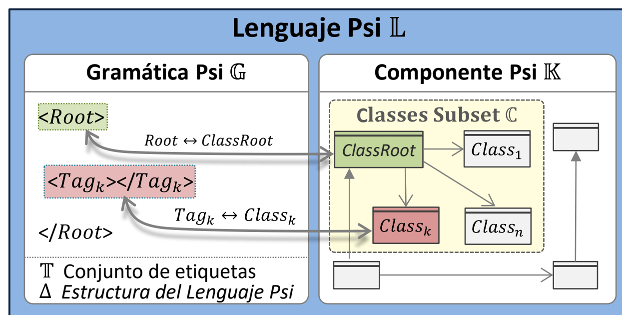


Figura 3-4. Definición de Lenguaje Psi $\mathbb{L}$, Gramática Psi $\mathbb{G}$, Componente Psi $\mathbb{K}$ y asociaciones entre $\mathbb{T} \leftrightarrow \mathbb{C}$.

Toda clase asociada de $Class_k \in \mathbb{C}$ debe heredar de la clase abstracta *PsiElement* del PsiXML. Esta clase abstracta implementa el comportamiento abstracto para cada elemento del lenguaje Psi. En particular, contiene el elemento DOM (referencia a la etiqueta Tag_k), contiene la lista de elementos hijos y el elemento padre, gestiona la estructura gramatical, realiza la validación de atributos y hace la evaluación de código *inline* JavaScript. Además, la clase abstracta *PsiElement* implementa dos lenguajes disponibles para cualquier etiqueta Tag_k:

M1. **Lenguaje PsiLI** (*Psi Link Information*). Es un lenguaje *inline* encargado de obtener información XML y JSON para la clase asociada $Class_k$, procesando los atributos *psi-context* (obtener información desde el *Contexto*) y *psi-document* (obtener información desde PsiData) de la etiqueta Tag_k. Su especificación es la siguiente:

$\langle data \rangle$ ::= { $\langle alias \rangle$ ":" $\langle key \rangle$ ":" $\langle selector \rangle$ ["$$"] }

donde $\langle alias \rangle$ es el alias de referencia en la que se encuentra la información en el *Contexto* o en PsiData, $\langle key \rangle$ es la clave para guardar la información y $\langle selector \rangle$ busca la información y la asocia usando el lenguaje de selección de jQuery. El token "$$" es un separador para especificar múltiples fuentes de información.

M2. **Lenguaje PsiCA** (*Psi Code Attribute*). Es el encargado de evaluar código *inline* en los atributos de cualquier etiqueta Tag_k a través de un conjunto de comandos de línea. Además, obtiene datos de la información asociada con PsiCA para establecer los valores de un atributo en la etiqueta Tag_k. Su especificación es la siguiente:

$\langle value \rangle$ ::= $\langle command \rangle$ $\langle line \rangle$
$\langle command \rangle$ ::= ("$>" | "$=" | "$?" | "$@")

Donde se tienen los siguientes comandos:

- *Comando* “$>”: crea una función JavaScript donde $\langle line \rangle$ es el cuerpo de la función y tiene como parámetros *self* (instancia de la clase asociada) y *context* (el *Contexto*).
- *Comando* “$=”: evalúa el código $\langle line \rangle$ y tiene como parámetros *self* y *context*.
- *Comando* “$?”: hace una llamada a una función de PsiData, donde $\langle line \rangle$ tiene el nombre de la función con su lista de los argumentos.
- *Comando* “$@”: obtiene el valor definido de la información asociada, donde $\langle line \rangle = \langle key \rangle.\langle attr \rangle$ es la clave y el nombre del atributo.

En términos generales, el valor $\langle \mathrm{line} \rangle$ antes de ser evaluado por un comando, se buscan y se cambian todos los valores “{{$\langle \mathrm{key} \rangle.\langle \mathrm{attr} \rangle$}}” en la información asociada.

En el FRAGMENTO 3.7 se muestra la plantilla JavaScript para definir un componente Psi, incluye la parte del componente del FRAGMENTO 3.6 y se agrega la asociación de clases con las etiquetas.

FRAGMENTO 3.7. Plantilla JavaScript para definir un componente Psi.

```
var MyComponentPsi = (function () {
  // Gramática Psi
  // Conjunto de identificadores de la gramática Psi
  var ID_TAG_T1 = "Root", ID_TAG_T1 = "T1", …, ID_TAG_Tm = "Tm";
  // Estructura del lenguaje
  var ID_TAGS = {};
  ID_TAGS[ID_TAG_ROOT] = {TAG:"Root", CLASS:"ClassRoot", CHILDREN: …, MULTIPLICITY: …,
                          STRICT: …, VALIDATOR: "…"};
  ID_TAGS[ID_TAG_T1] = {TAG:"Tag1", CLASS:"Class1", CHILDREN: …, MULTIPLICITY: …,
                        STRICT: …, VALIDATOR: "…"};
  …
  ID_TAGS[ID_TAG_Tm] = {TAG:"Tagm", CLASS:"Classm", CHILDREN: …, MULTIPLICITY: …,
                        STRICT: …, VALIDATOR: "…"};

  //definición de la gramática Psi
  var GRAMMAR = {
    NAME: "GrammarName",
    TAGS: ID_TAGS,
    ROOT: ID_TAG_ROOT // Etiqueta raíz. Para algún identificador ID_TAG_ROOT
  };

  // Clases asociadas a la gramática Psi
  function ClassRoot(){}; CLassRoot.methods({...}); // Implementación de clase ClassRoot
  function Class1(){}; Class1.methods({...}); // Implementación de clase Class1
  ...
  function Classm(){}; C1.methods({...}); // Implementación de clase Classm

  // Elementos de programación adicionales al componente Psi
  return { //Interfaz del componente Psi };
})();
```

3.4 METODOLOGÍA PARA EL DISEÑO E IMPLEMENTACIÓN DE LENGUAJES PSI

La construcción de un XML-DSL en el cliente web puede llegar a ser una labor dispendiosa en términos de programación, pero también es difícil modelar la gramática para lograr una cohesión armónica entre la gramática y su funcionalidad. La construcción de XML-DSLs mediante la definición de lenguajes Psi se basa en el uso de las plantillas del FRAGMENTO 3.4, el FRAGMENTO 3.7 y la implementación dinámica de clases propuestas en el FRAGMENTO 3.2.

Para simplificar el diseño e implementación de un lenguaje Psi en PsiEngine, se resume en los siguientes pasos:

P1. *Análisis y diseño del lenguaje Psi*. Plantear la solución del problema como XML-DSL y crear un diseño o esquema preliminar del lenguaje.

P2. *Diseño del componente Psi*. Crear el diagrama de estructura del lenguaje Psi (PsiLSD), especificar los validadores de atributos para la gramática Psi (PsiGVA).

P3. *Diseño del componenete Psi*. Crear el diagrama de clases del componente Psi.

P4. *Implementación del componente Psi*. Apoyados en el análisis y diseño, el PsiLSD, el PsiGVA y el diagrama de clases implementar el componente Psi.

P5. *Pruebas del lenguaje Psi*. Realizar pruebas unitarias y funcionales del componente Psi y crear la documentación necesaria del lenguaje.

Cabe resaltar que, tanto para el caso de estudio lenguaje MiniPsi que se verá en la sección 3.7, como para el resto de lenguajes Psi que se desarrollaron en esta tesis, se siguieron estos pasos. La implementación de un lenguaje Psi es una tarea parcialmente automatizada por el modelo de programación Psi que se verá en detalle en el Capítulo 4.

3.5 EL INTÉRPRETE PSIXML

El *Intérprete PsiXML* (PsiXML) es la pieza central del PsiEngine y la sección está dedicada a explicar su funcionamiento y resumir su implementación. La Figura 3-5 muestra el diagrama de bloques de PsiXML y está conformado por las siguientes partes funcionales:

F1. **Administrador de Lenguajes Psi**. Es el encargado de registrar y administrar los **lenguajes Psi**. Contiene los elementos de programación para definir, analizar y evaluar cada uno de los elementos de un programa Psi. Da soporte a la transformación y creación dinámica de objetos. Contiene la clase abstracta *PsiElement* descrita en la sección anterior.

F2. **Traductor** $S_k \rightarrow O_k$. Es el encargado de la transformación de *PsiCode* S_k escrita con una gramática Psi, a un *PsiObject* O_k. Para ello, usa un *Scanner* para realizar el análisis léxico XML del S_k, y un *Parser* para realizar el análisis sintáctico y semántico con la gramática Psi y obtener el objeto O_k. Estos análisis consisten en recorrer en forma de árbol, validar el elemento, y asociar y ejecutar la funcionalidad del componente Psi correspondiente.

F3. **Evaluador**. Es el encargado de *evaluar* un *PsiCode* S_k dentro del PsiXML para obtener un *PsiObject* O_k. El término *evaluar* en PsiXML consiste en:

   i. Crear un objeto **PsiProgram** $\boldsymbol{P_k}$ y añadirlo al administrador de programas Psi.

   ii. Traer el *PsiCode* S_k y los *Recursos* vía AJAX, e iniciar el *Contexto*.

   iii. Obtener la definición del lenguaje Psi $\mathbb{L}$ del administrador de lenguajes Psi.

   iv. Realizar la transformación de *PsiCode* S_k con la gramática Psi obtenida en *iii.*, y hacer uso del **Traductor** $S_k \rightarrow O_k$ para obtener *PsiObject* O_k y la *Tabla de Objetos* $\mathbb{Y}$.

   v. Reportar en caso de error los mensajes y las advertencias respectivas.

F4. **Administrador de Programas Psi**. El encargado de administrar los programas Psi que se han evaluado sobre el PsiXML.

Como caja negra, PsiXML toma el código fuente *PsiCode*, el *Contexto* y los *Recursos* como elementos de entrada. A continuación, se hace la transformación de *PsiCode* con un lenguaje Psi previamente registrado y se obtiene como resultado un *PsiObject* interpretado y/o un conjunto de *Errores y Advertencias*, si los hay, para dar comienzo a su uso. PsiXML tiene la funcionalidad de evaluar múltiples programas ($\boldsymbol{P_1}, \dots, \boldsymbol{P_n}$) escritos en diferentes lenguajes Psi con la capacidad de intercambio de información, de funciones y de objetos en el área común **PsiData**.

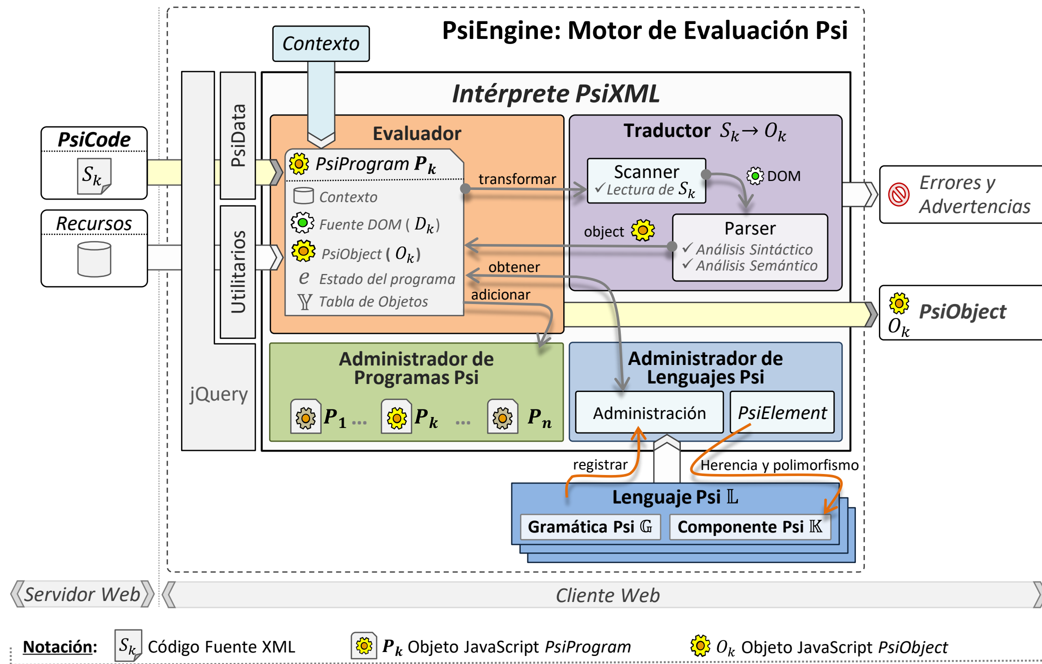


Figura 3-5. PsiEngine: Diagrama de bloques del *Intérprete PsiXML*

3.5.1 Formalización del Intérprete

Para la Figura 3-5, el conjunto $\Sigma = \bigcup_{i=1}^{n} \mathbb{K}_i$ representa todos los componentes disponibles para la aplicación en un CSWE. En particular, se tomarán $\Sigma^* = \bigcup_{i=1}^{m} \mathbb{K}_i \subseteq \Sigma$, con $m < n$, como el subconjunto de componentes Psi que dan soporte a los lenguajes Psi $\Psi = \{\mathbb{L}_1, \mathbb{L}_2, \dots, \mathbb{L}_m\}$, respectivamente, tal que, para cada lenguaje Psi $\mathbb{L}_i = \langle \mathbb{G}_i | \mathbb{K}_i | \mathbb{T}_i \leftrightarrow \mathbb{C}_i \rangle, i = 1, \dots, m$, se define con la tupla (3-4), donde $\mathbb{G}_i$ es la gramática Psi, $\mathbb{K}_i \in \Sigma^*$, $\mathbb{T}_i$ el conjunto de etiquetas con sus respectivas clases asociadas $\mathbb{C}_i$. Además, se denotará por $\mathbb{X}$ el *Contexto* y por $\mathbb{R}$ los *Recursos* de PsiEngine.

En particular, un *PsiCode* S_k^i es un fichero XML S_k escrito con la gramática $\mathbb{G}_i$, y el $\mathbb{S}^i = \{S_1^i, \dots, S_k^i, \dots, S_n^i\}$ es el conjunto de todos los códigos fuentes disponibles del lenguaje $\mathbb{L}_i$ para ser evaluados por el PsiXML. En consecuencia, un programa Psi $\boldsymbol{P}_k^i$ puede ser visto como pasos de transformación:

$$\boldsymbol{P}_k^i = \{S_k^i, \mathbb{X}, \mathbb{R}\} \xrightarrow{Scanner} \{\langle D_k^i | \mathbb{L}_i \rangle, \mathbb{X}, \mathbb{R}\} \xrightarrow{Parser} \{\mathbb{Y}, O_k^i, M_k^i, \mathbb{X}^*, \mathbb{R}^*\}, \tag{3-5}$$

donde D_k^i es el documento DOM obtenido a partir de S_k^i al aplicar el *Scanner*. El objeto O_k^i es el resultado *PsiObject*, obtenido a partir del *Parser*, o en su defecto, se obtienen los mensajes de errores y advertencias M_k^i. El objeto $\mathbb{Y}_k^i = \left\{ oC_{i_r}^j \in C_{i_r} | C_{i_r} \in \mathbb{C}_i, r = 1, \ldots n, j \text{ } es \text{ } un \text{ } \boldsymbol{UID}^8 \right\}$ representa la *Tabla de Objetos* del programa $\mathbf{P}_\mathrm{k}^\mathrm{i}$, donde $oC_{i_r}^j$ es la instancia *j*-ésima de la clase C_{i_r} del conjunto de clases asociadas $\mathbb{C}_i$.

Por lo general, la evaluación de un programa $\boldsymbol{P}_k^i$ implica cambios tanto en el contexto (denotado por $\mathbb{X}^*$) como en los recursos (denotado por $\mathbb{R}^*$). Además, un programa $\boldsymbol{P}_k^i$ se rige por el diagrama de estados de la Figura 3-6, donde cada estado está definido como:

E1. INIT: indica que el programa $\boldsymbol{P}_k^i$ da inicio y tiene a disposición $\{S_k^i, \mathbb{X}, \mathbb{R}\}$.

[8] UID es el identificador único de instancia de clase para oC_k^i.

E2. SCANNER: indica que se ha realiza el análisis léxico y se obtiene $\{D_k^i, \mathbb{X}, \mathbb{R}\}$.

E3. PARSE: indica que se ha realiza el análisis sintáctico y semántico de D_k^i basado en el lenguaje Psi $\mathbb{L}_i$ para obtener la *Tabla de Objetos* $\mathbb{Y}$ y el *PsiObject* O_k^i.

E4. RUN: indica que el programa $\boldsymbol{P}_k^i$ está en ejecución de forma exitosa y devuelve el objeto $\{O_k^i, \mathbb{X}^*, \mathbb{R}^*\}$.

E5. ERROR: indica que el programa $\boldsymbol{P}_k^i$ ha terminado con error.

E6. STOP: indica que el programa $\boldsymbol{P}_k^i$ se ha detenido.

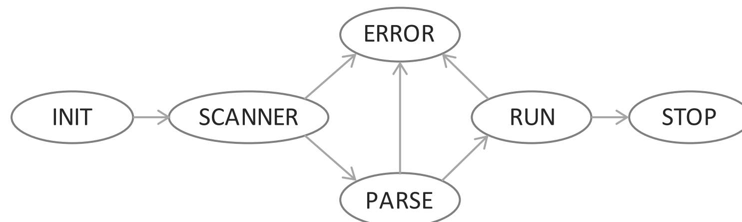


Figura 3-6. Diagrama de estados para un programa Psi.

En consecuencia, se puede expresar el PsiXML denotado por Ψ_{XML} como una transformación:

$$\{S_k^i, \mathbb{X}, \mathbb{R}\} \xrightarrow{\Psi_{XML}\langle\Sigma\rangle} \{O_k^i, \mathbb{X}^*, \mathbb{R}^*\}, \tag{3-6}$$

donde Σ es el conjunto de componentes disponibles, incluidos en los componentes Σ^*, S_k^i es un código fuente escrito en un lenguaje $\mathbb{L}_i$, y se obtiene como resultado un objeto O_k^i que da solución a un problema específico de una aplicación web, o parte de ella.

Sin pérdida de generalidad, se puede expresar que la solución a un problema específico de una aplicación web, puede ser formalizada como «*la evaluación de un conjunto de programas Psi sobre el Intérprete PsiXML*». Es decir:

$$\bigcup_{i=1..m} \bigcup_{k=0..t_i} \{S_k^i, \mathbb{X}, \mathbb{R}\} \xrightarrow{\Psi_{XML}\langle\Sigma\rangle} \bigcup_{i=1..m} \bigcup_{k=0..t_i} \{O_k^i, \mathbb{X}^*, \mathbb{R}^*\}, \tag{3-7}$$

donde t_i es un entero positivo para cada $i = 1..m$.

3.5.2 Implementación del Intérprete

En la Figura 3-7 se muestra el diagrama de clases simplificado correspondiente al **Componente PsiXML**, que implementa la funcionalidad del Intérprete PsiXML de PsiEngine formalizada en la sección anterior. Está sección esta dedica a explicar su implementación.

En el Componente PsiXML las partes funcionales del diagrama de bloques de la Figura 3-5 están divididas de la siguiente forma: **Definición de Lenguajes Psi**, contiene el subcomponente LanguagePsi y la clase *PsiElement*; el **Traductor** $S_k^i \to O_k^i$, contiene las clases *Parser* y *Scanner*; el **Evaluador**, contiene las clases *Program* y *Trash*, y el tipo enumerativo *ProgramState*; y **Administrador de Programas Psi**, contiene el objeto *Programs*.

El subcomponente **LanguagePsi** tiene dos funcionalidades importantes: (i) registrar y administrar los lenguajes Psi $\mathbb{L}$, y (ii) manipular los elementos de cualquier lenguaje Psi $\mathbb{L}_k$ mediante su clase abstracta *PsiElement*. El objeto *Validator*, implementa funciones de validación de atributos de PsiGVA para una gramática $\mathbb{G}_k$ y tiene la capacidad de expandirse. Las clases *DefineAttribute*, *DefinitionElement* y *DefinitionLanguage*, implementan la estructura gramatical de un lenguaje Psi $\mathbb{L}_i$ para realizar el análisis sintáctico y semántico para un código fuente Psi S_k^i.

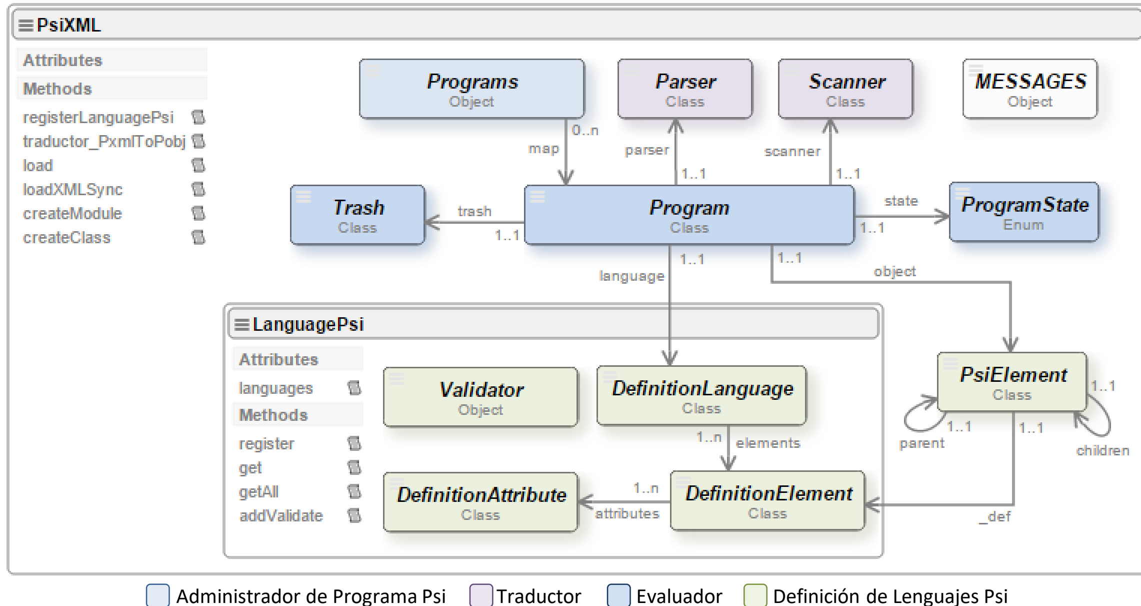


Figura 3-7. Diagrama de clases del *Intérprete PsiXML* (Componente PsiXML)

La clase abstracta ***PsiElement*** es la piedra angular de un lenguaje Psi, ya que cualquier elemento de un lenguaje Psi debe heredar de ella. La clase *PsiElement* mantiene la referencia a la definición del elemento Psi (referencia a una instancia de DefinitionElement), la referencia al elemento DOM del código fuente Psi, la lista de sus elementos hijos y la referencia a su elemento padre. Además, tiene la capacidad de procesar los lenguajes **PsiLI** (Psi Link Information) y **PsiCA** (Psi Code Attribute) definidos en la sección 3.3.2.

La clase *Scanner* realiza el análisis léxico, consiste en la carga y validación de un documento XML, tarea relegada al cliente web. Hay dos formas de cargar un código fuente Psi: a través (i) una conexión AJAX, o (ii) creando en el cliente un documento XML desde una cadena de caracteres. Ambas formas retornan el código fuente DOM D_k^i. En caso de errores de comunicación o errores en el documento XML, devuelve una excepción.

La clase *Parser* es la encargada de implementar el análisis léxico y análisis sintáctico del lenguaje Psi $\mathbb{L}_i$. Busca la definición del lenguaje $\mathbb{L}_i$ en el componente LanguagePsi y obtiene su definición. Se recorren los elementos DOM del código fuente D_k^i en forma de árbol y se valida la estructura Δ_i de la gramática $\mathbb{G}_i$. Se obtiene la etiqueta $tag_k^i \in \mathbb{T}_i$ y con ella se busca la definición del elemento para hacer la instancia de la clase asociada $C_k^i \in \mathbb{C}_i$, ejecutar su código asociado, y registrarlo en la *Tabla de Objetos* (análisis sintáctico). Además, se comprueba la validez semántica, verificando si cada elemento con sus atributos es aceptado por la gramática $\mathbb{G}_i$ (análisis semántico). En caso contrario, es enviada a la papelera del programa (clase *Trash*) y/o se genera una excepción si el elemento es necesario.

El tipo enumerativo *ProgramState* define los estados de un programa Psi y la clase *Program* es la encargado de *evaluar* (método *execute*) el código fuente Psi S_k^i para obtener un programa Psi $\boldsymbol{P}_k^i$, cuyo resultado es un objeto Psi O_k^i. Está evaluación implementa la lógica del diagrama de estados de la Figura 3-6.

Por último, los principales métodos que tiene a disposición el Componente PsiXML se describen a continuación:

M1. Método ***registerLanguagePsi***: registra un lenguaje Psi con el *alias* (nombre al lenguaje Psi $\mathbb{L}_i$), la estructura de la gramática Psi $\mathbb{G}_i$ y el nombre del parser, en el subcomponente LanguagePsi para

su uso.

M2. Método ***traductor_PxmlToPobj***: implementa la lógica de traducción $S_k^i \rightarrow O_k^i$ de un *PsiCode* a un *PsiObject*. Al iniciar el proceso de traducción, valida que el nombre del programa que no esté registrado, seguido obtiene el lenguaje Psi $\mathbb{L}_i$ mediante su *alias*, luego se crea una instancia $\boldsymbol{P}_k^i$ de la clase *Program* con el nombre, el lenguaje, las opciones, y se da inicio a su evaluación. Si es exitosa la ejecución, se registra en *Programs* y se devuelve el objeto Psi O_k^i desde el programa.

M3. Método ***loadXMLSync***: obtiene un documento XML vía AJAX de forma sincrónica.

M4. Método ***load***: obtiene un documento (XML, JSON, CSS, HTML, etc.) vía AJAX.

M5. Método ***createModule***: crea un módulo de programación como objeto estático JavaScript.

M6. Método ***createClass***: implementa una clase basado en el FRAGMENTO 3.2.

3.6 IMPLEMENTACIÓN DEL PSIENGINE

En la Figura 3-8 se muestra la captura de la página inicial de hilas.ii.uam.es/api, para la documentación de implementación del API de PsiEngine (el código fuente se encuentra en el CD adjunto y en http://github.com/echavarriaga/PsiEngine). El diagrama de clases detallado de PsiEngine es un diagrama interactivo donde cada elemento de programación se puede ver la documentación en línea y el código de implementación (opción de menú “Class Diagram”). También dispone de la documentación del componente PsiXML (opción “PsiXML Interpreter”) y la documentación del subcomponente LanguagePsi (opción “Psi Languages”). La documentación del resto de elementos de programación (PsiData y Utilitarios) se encuentra en la opción “Additional”. Cabe resaltar que parte de ésta documentación es generada automáticamente con el uso de plantillas XSL por el PsiModel del Capítulo 4.

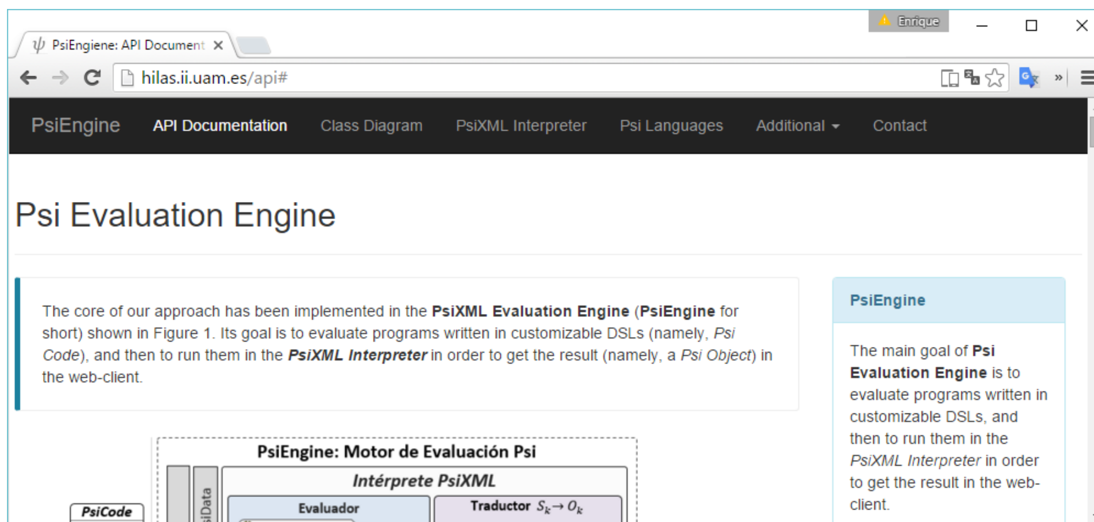


Figura 3-8. Documentación del API del Motor de Evaluación Psi (PsiEngine).

En la Figura 3-9 se muestra el diagrama de clases y los componentes principales de PsiEngine. Este diagrama lo conforman el componente jQuery, el objeto *PsiData*, el componente **PsiXML** y los **Utilitarios**.

El objeto *PsiData* tiene como objetivo clasificar ítems según las disposiciones y necesidades de los lenguajes Psi. Por defecto, administra las variables de contexto, los documentos XML, los ficheros JSON y las funciones JavaScript que son compartidas por todos los programas Psi.

Los **Utilitarios** están conformados por el componente **PsiText** y las clases de errores y advertencias (*PsiOut*, *PsiError* y *PsiWarning*). El componente **PsiText** administra los textos informativos, los de error y los de advertencia, para el PsiXML y los lenguajes Psi y componente creado para el PsiEngine. Los

conjuntos de textos son ordenados por idioma y almacenados con un alias y su idioma se configura en el componente Text. Los textos son utilizados en los diferentes objetos, clases y componentes cuando hay un error o una advertencia, según corresponda.

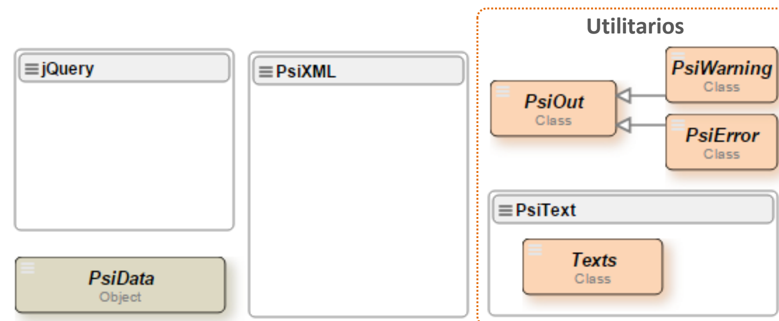


Figura 3-9. Diagrama de clases y componentes de Motor de Evaluación Psi (PsiEngine).

En la Figura 3-1, los *Errores y Advertencias* se implementan en PsiEngine como clases: *PsiOut*, *PsiError* y *PsiWarning*. La clase *PsiOut* es la encargada de construir el mensaje con la ayuda de PsiText, y la capacidad de guardarlos, sacarlos por consola y/o re-direccionarlos a un servidor de log con el objetivo de su posterior consulta. Las clases *PsiError* y *PsiWarning* heredan de *PsiOut* y se diferencian del tipo de mensaje.

3.7 Caso de estudio: Lenguaje MiniPsi

Como ejemplo ilustrativo de creación de un Lenguaje Psi se diseñará e implementará el **Lenguaje MiniPsi,** y se creará la aplicación web "MiniPsi Language" (entorno para la creación y evaluación de programas MiniPsi) para validar el lenguaje MiniPsi. Cada subsección es un paso de la metodología propuesta en la sección 3.4 para el diseño e implementación del lenguaje MiniPsi.

3.7.1 Análisis y diseño del Lenguaje MiniPsi

El lenguaje MiniPsi es un XML-DSL para la creación de variables, funciones, clases, instancias de clase y bloques de código JavaScript, dentro del CSWE. Es decir, MiniPsi es un lenguaje de programación ligero que combina XML, lenguaje JavaScript y las cualidades nativas de los lenguajes Psi.

En el Fragmento 3.8 se muestra como ejemplo un programa con nombre "hello-program" especificado en lenguaje MiniPsi. El programa se inicia definiendo una variable "hello" con una cadena de caracteres asociada. A continuación define una función "Great" con un argumento *message*, y el código JavaScript asociado a la función es *alert(message)*. Por último, el programa especifica un bloque de código JavaScript llamando a la función "Greet" y pasando como parámetro la variable "hello".

Fragmento 3.8. Programa en lenguaje MiniPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<MiniProgram name="hello-program">
  <Var name="hello">"Hola Mundo!!"</Var>
  <Function name="Greet" arguments="message">alert(message);</Function>
  <Block>Greet(hello);</Block>
</MiniProgram>
```

3.7.2 Diseño del Lenguaje MiniPsi y del Componente MiniPsi

En la Figura 3-10(a) se muestra el PsiLSD y en la Figura 3-10(b) se muestra el PsiGVA del lenguaje MiniPsi.

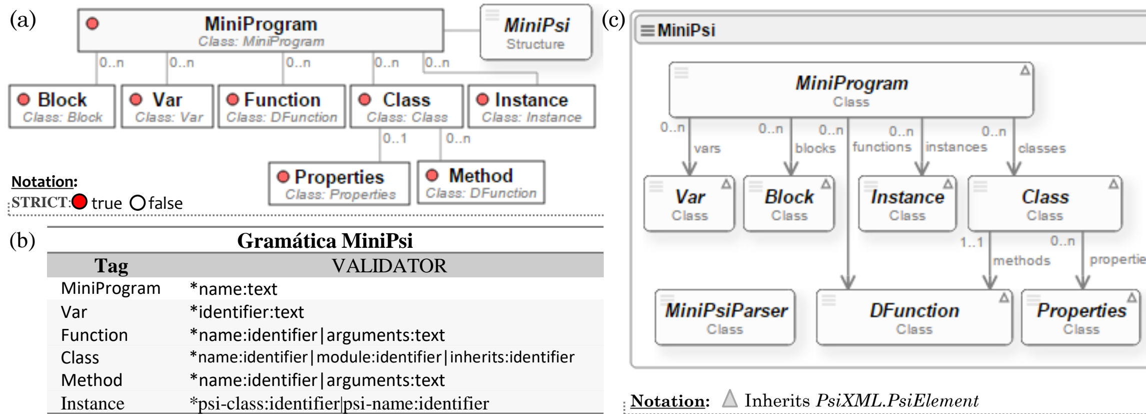


Figura 3-10. (a) PsiLSD del lenguaje MiniPsi. (b) PsiGVA del lenguaje MiniPsi. (c) Diagrama de clases para el componente MiniPsi.

El MiniPsi define las siguientes etiquetas: **MiniProgram** (etiqueta raíz, inicio del programa), **Var** (variable), **Function** (función), **Class** (clase), **Properties** (propiedades de la clase), **Instance** (instancia de clase), **Method** (método de la clase) y **Block** (bloque de ejecución de JavaScript). En la Figura 3-10(c) se presenta el diagrama de clases simplificado para el **componente MiniPsi**, y en la Tabla 3-1 se resume la asociación entre clases y etiquetas, así como la funcionalidad de las clases del componente MiniPsi.

Tabla 3-1. Etiquetas y clases asociadas del lenguaje MiniPsi.

Etiqueta	Clase	Descripción de la clase
MiniProgram* (1)	*MiniProgram**	Define el cuerpo del programa con un nombre (atributo *name*).
Var (n)	*Var*	Define el comportamiento asociado a la variable.
Function (n), **Method** (n)	*DFunction*	Define la funcionalidad relacionada para la creación de una función o un método de una clase.
Block (n)	*Block*	Tiene la responsabilidad de ejecutar el código JavaScript que figura dentro de la etiqueta **Block**
Class (n)	*Class*	Define una clase en JavaScript, donde el nombre (atributo *name*) es obligatorio, puede estar definido en un módulo (atributo *module*) y tener herencia (atributo *inherits*), además tiene propiedades (una instancia de la clase *Properties*) y métodos (varias instancias de la clase *DFunction*).
Propiedades (1)	*Properties*	Gestiona las propiedades de una clase JavaScript.
Instance (n)	*Instance*	Crea una instancia de una clase definida.

Nota: *: etiqueta o clase raíz; (1) Simple; (n) Múltiple.

3.7.3 Implementación del Componente MiniPsi

En la URL hilas.ii.uam.es/minipsi (el código fuente se encuentra en el CD adjunto y en http://github.com/echavarriaga/MiniPsi) se muestra en detalle el diagrama de clases del **Componente MiniPsi**, con la información de implementación de cada elemento de programación que la conforma.

Usando la plantilla JavaScript definida en Fragmento 3.7 para la creación de componentes Psi, en el Fragmento 3.9 se muestra el esqueleto de implementación JavaScript para el Componente MiniPsi.

Fragmento 3.9. Esqueleto de implementación JavaScript del componente MiniPsi.

```
var MiniPsi = (function () {
  // Gramática Psi

  // Conjunto de etiquetas para la gramática MiniPsi
  var ID_TAG_PROGRAM = "mini_program", ID_TAG_VAR = "var", ID_TAG_FUNCTION = "function",
      ID_TAG_BLOCK = "block", ID_TAG_CLASS = "class", ID_TAG_PROPERTIES = "properties",
      ID_TAG_METHOD = "method", ID_TAG_INSTANCE = "instance";
```

```
    // Estructura de la gramática MiniPsi
   var ID_TAGS = {};
   ID_TAGS[ID_TAG_PROGRAM] = {
     TAG: "Program", CLASS:"MiniProgram", MULTIPLICITY: "0..1", VALIDATOR: "*name:text",
     CHILDREN: {
       "Var": ID_TAG_PROGRAM, "Function": ID_TAG_FUNCTION,
       "Block": ID_TAG_BLOCK, "Class": ID_TAG_CLASS
     }
   };
   ID_TAGS[ID_TAG_VAR] = {
     TAG: "Var", CLASS:"MiniProgram", MULTIPLICITY: "0..n", VALIDATOR: "*identifier:text"
   };
   ID_TAGS[ID_TAG_FUNCTION] = {
     TAG: "Function", CLASS:"DFunction", MULTIPLICITY: "0..n",
     VALIDATOR: "*name:identifier|arguments:text"
   };
   ID_TAGS[ID_TAG_BLOCK] = { TAG: "Block", CLASS:"Block", MULTIPLICITY: "0..n" };
   ID_TAGS[ID_TAG_CLASS] = {
     TAG: "Class", CLASS:"Class", MULTIPLICITY: "0..n",
     VALIDATOR: "*name:identifier|module:identifier|inherits:identifier",
     CHILDREN: { "Properties": ID_TAG_PROPERTIES, "Method": ID_TAG_METHOD },
   };
   ID_TAGS[ID_TAG_PROPERTIES] = { TAG: "Properties", CLASS:"Properties", MULTIPLICITY: "0..1" };
   ID_TAGS[ID_TAG_METHOD] = {
     TAG: "Method", CLASS:"DFunction", MULTIPLICITY: "0..n",
     VALIDATOR: "*name:identifier|arguments:text"
   };
   ID_TAGS[ID_TAG_INSTANCE] = {
     TAG: "Instance", MULTIPLICITY: "0..n",
     VALIDATOR: "*psi-class:identifier|psi-name:identifier"
   };

   // Definición de la gramática MiniPsi
   var GRAMMAR = { NAME: "MiniPsi", TAGS: ID_TAGS, ROOT: ID_TAG_PROGRAM };
   //Classes implementation
   function MiniProgram(){}; MiniProgram.methods({...}); // MiniProgram Class
   function Var(){}; Var.methods({...}); // Var Class
   function DFunction(){}; DFunction.methods({...}); // Function Class
   function Block(){}; Block.methods({...}); // Block Class
   function Class(){}; Class.methods({...}); // Class Class
   function Properties(){}; Properties.methods({...}); // Properties Class
   function Instance(){}; Properties.methods({...}); // Instance Class

   // Elementos de programación adicionales del componente Psi
   return {
     //Interfaz del componente Psi
     MiniProgram: MiniProgram, Var: Var, DFunction: DFunction, Block: Block, Class: Class,
     Properties: Properties, Instance: Instance
     //Elementos adicionales de programación del componente Psi
   };
 })();
```

3.7.4 Ejemplos de uso del lenguaje MiniPsi

Para comprender mejor los conceptos explicados en el capítulo, se aportan dos ejemplos del uso de los lenguajes PsiLI y PsiCA. Primero se define la información de personas en formato XML, como se muestra en el Fragmento 3.10.

Fragmento 3.10. Fichero XML “people.xml”.

```
<?xml version="1.0" encoding="utf-8"?>
<People>
  <Person id="p1" first="Luke" last="Skywalker" age="25"/>
  <Person id="p2" first="Obi-Wan" last="Kenobi" age="45"/>
</People>
```

Segundo, se define el *Contexto* e incluye la información personas (ver FRAGMENTO 3.11). Esta información es cargada vía AJAX de forma sincrónica y almacenada en el objeto *context*. Por último, el uso de PsiLI y PsiCA se un programa MiniPsi (ver FRAGMENTO 3.12) y el diseño de patrones DAO con MiniPsi (ver FRAGMENTO 3.13).

FRAGMENTO 3.11. Contexto *people* para el fichero "people.xml".

```
var context = {
  people: PsiXML.loadXMLSync("people.xml")
}
```

El FRAGMENTO 3.12 se define un programa MiniPsi con dos variables y un bloque de código JavaScript. En la primera etiqueta **Var** se establece en el atributo *psi-context* la referencia al campo *people* (elemento DOM) del *Contexto*, con el alias *info* en la instancia de la clase asociada *Var*. Está referencia se consigue buscando "`Person[id=p1]`" y haciendo uso de los selectores de jQuery (ejecución del PsiLI). Esto implica que al aplicar el PsiCA a la etiqueta **Var**, la variable *person* concatena el nombre y apellido (value="`Luke Skywalker`") usando el comando "`$=`" y la información de *info*. Para la segunda etiqueta **Var** se obtiene un valor de `25` y se almacena en la variable *age*. En consecuencia, que con la etiqueta **Block** se muestra el mensaje "`Luke Skywalker has 25 years`".

FRAGMENTO 3.12. Usando los lenguajes PsiLI y PsiCA en un programa MiniPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<MiniProgram name="person-program">
  <Var name="person" psi-context="people:info=Person[id=p1]"
       value="$='{{info.first}} {{info.last}}'"/>
  <Var name="age" psi-context="people:info=Person[id=p1]" value="$@info.age"/>
  <Block>alert(person+" has "+age+" years");</Block>
</MiniProgram>
```

En el FRAGMENTO 3.13 se muestra una aplicación diferente de PsiLI y PsiCA en el lenguaje MiniPsi. En este fragmento se especifica el acceso a los datos de un archivo XML utilizando el patrón de diseño de Objeto de Acceso a Datos (con su sigla en inglés DAO, *Data Access Object*), de modo que las propiedades de la clase se pueden obtener directamente de las fuentes de información XML.

Por lo tanto, en el FRAGMENTO 3.13 se muestra la clase *Person* (etiqueta **Class**) que especifica el acceso a la información del *Contexto* del ejemplo anterior. Con la etiqueta **Instance**, se crea una instancia de la clase *Person* obteniendo la información de la persona con `id=p2`. En consecuencia, con la etiqueta **Block**, se obtendría el mensaje "`Hi Obi-wan Kenobi!!`".

FRAGMENTO 3.13. Usando los lenguajes PsiLI y PsiCA para un diseño de patrones DAO.

```
<?xml version="1.0" encoding="utf-8"?>
<MiniProgram name="person-class" module="MyModule">
  <Class name="Person" arguments="first, last" module="Users">
    <Properties first="first|s:@first" last="last|s:@last"
                age="null|i:@age" alias="null|s:@alias"/>
    <Method name="fullName" arguments="">return this.first+" "+this.last;</Method>
  </Class>
  <Instance psi-name="p2" psi-class="Users.Person"
            data-context="people:info=Person[id=p2]" psi-key="info"/>
  <Block>alert("Hi "+p2.fullName()+"!!");</Block>
</MiniProgram>
```

3.7.5 APLICACIÓN WEB "MINIPSI LANGUAGE"

En la Figura 3-11 se muestra la página web "MiniPsi Language" que se ha desarrollado para el lenguaje MiniPsi. Esta página es un entorno ligero para la edición y ejecución de programas escritos en lenguaje MiniPsi. En hilas.ii.uam.es/minipsi/examples se pueden explorar diversos ejemplos, incluyendo los

fragmentos de esta sección. Además, el código fuente se encuentra disponible en el CD adjunto y en http://github.com/echavarriaga/MiniPsi.

La página citada permite la creación y ejecución de programas MiniPsi, así como, la edición del *Contexto*. Además posee una salida de texto, una salida gráfica y la ayuda en línea. En la pestaña "Files" implementa tres grupos de ejemplos: (i) ejemplos básicos (definición de variable, función, clase e instancia), (ii) asociación de información XML (en variable, función e instancia) y (iii) definición de figuras.

En la definición de figuras se crea un conjunto de clases para formar una librería gráfica ligera basada en el canvas HTML. Para crear figuras, primero se debe ejecutar el programa de definición. En la Figura 3-11 se muestra precisamente el uso de estas definiciones.

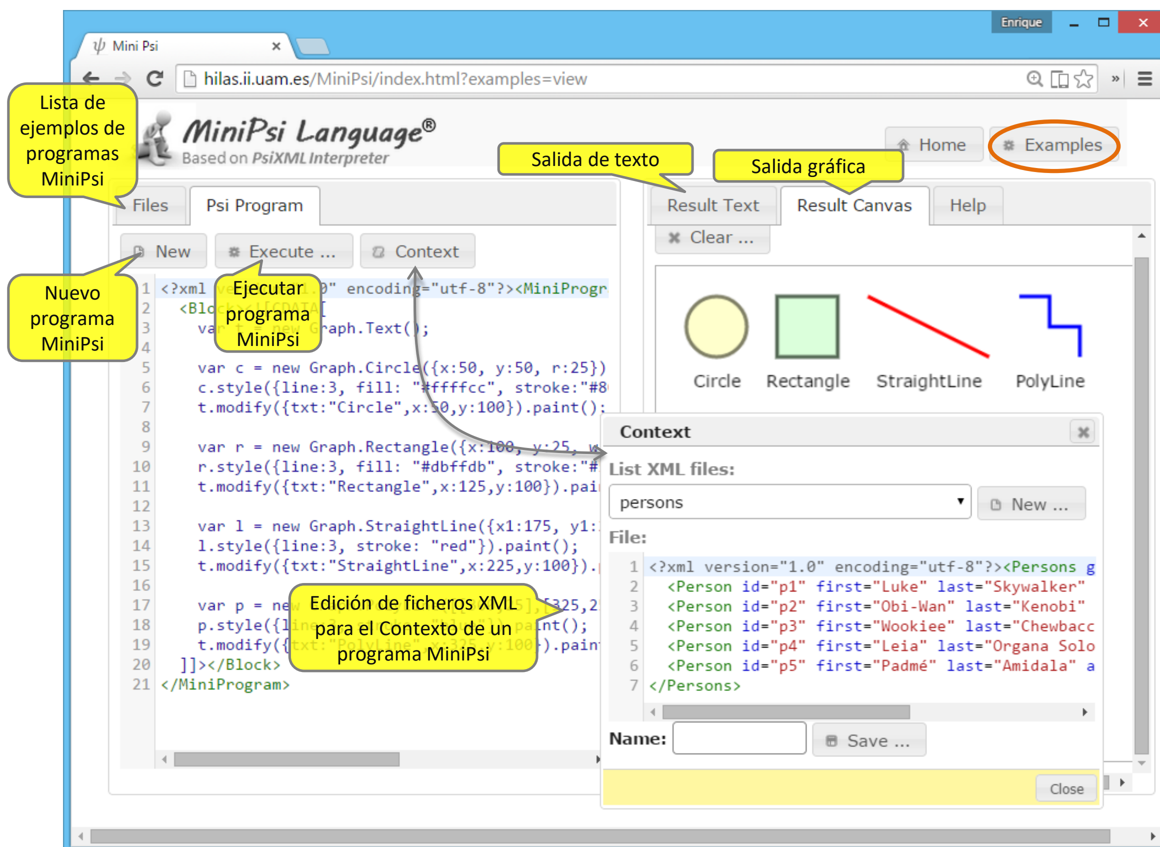


Figura 3-11. Aplicación Web MiniPsi: ejemplos de programación en lenguaje MiniPsi.

3.8 Resumen

Los DSLs permiten crear lenguajes especializados de programación que ofrecen abstracciones de alto nivel y técnicas para la construcción de sistemas de información enfocadas al dominio de un problema. Su objetivo es permitir a los expertos del dominio realizar las tareas de manera más eficiente, produciendo sistemas de mayor calidad y fiabilidad. En un CSWE, los XML-DSLs son lenguajes fáciles de entender y procesarse como documento DOM, y pueden agregar funcionalidad a las aplicaciones web.

En este capítulo se ha definido y desarrollado el Motor de Interpretación y Evaluación de Soluciones Programables o **Motor de Evaluación Psi** (PsiEngine, Programmable Solutions Interpreter Engine). El PsiEngine combina los lenguajes HTML5, CSS3, JavaScript y DOM, con las tecnologías, servicios y herramientas de la Web 2.0, y con la definición de XML-DSLs, para la construcción de componentes web, widgets web y/o páginas web dinámicas que dan solución a problemas específicos dados en una

aplicación web, o partes de ella. En la Figura 3-12 se resume en un diagrama de bloques, la creación de soluciones a problemas específicos basados en PsiEngine para un CSWE.

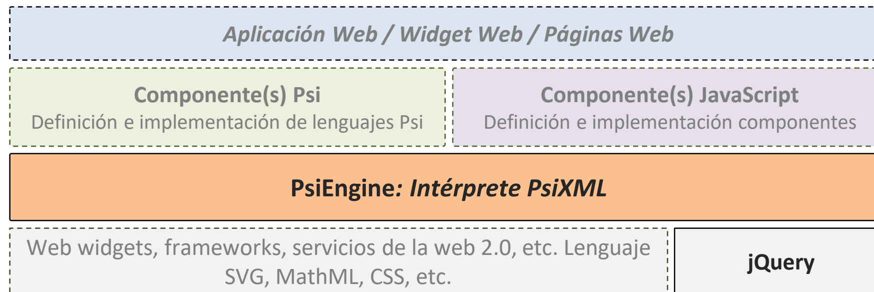


Figura 3-12. Diagrama de bloques para la creación de soluciones Psi basados en PsiEngine en un CSWE.

El principal objetivo del PsiEngine es interpretar y evaluar programas escritos en un Lenguaje Psi en un CSWE, cubriendo el objetivo de investigación O1 planteado para esta tesis. Un Lenguaje Psi es un XML-DSL que tiene una gramática XML y una funcionalidad asociada, con capacidad de relacionar información heterogénea (XML/JSON) y de ejecutar código *inline* JavaScript. El encargado de interpretar y evaluar los Lenguajes Psi, es el Intérprete PsiXML (PsiXML, pieza central del PsiEngine). Este intérprete es un *framework* JavaScript liviano, genérico (funciona en cualquier navegador de cualquier dispositivo), y usa jQuery (jquery.com) como parte de su implementación.

Adicionalmente, en este capítulo se establecieron los pasos de la metodología para el diseño e implementación de lenguajes Psi y se creó el caso de estudio Lenguaje MiniPsi aplicando esta metodología, para validar el PsiEngine.

Para concluir el capítulo, se desarrolló el caso de estudio denominado lenguaje MiniPsi. Este lenguaje (implementado en el Componente MiniPsi) es un XML-DSL que define variables, funciones, bloques de código JavaScript, clases e instancias de clase. En la Figura 3-13 se muestra el diagrama de bloques para la aplicación web "MiniPsi Language" cuyo objetivo es validar el lenguaje MiniPsi.

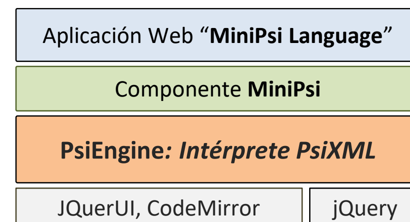


Figura 3-13. Diagrama de bloques de la aplicación web "MiniPsi Language".

Capítulo 4. PsiModel: Modelo de programación Psi para la generación de código JavaScript

Este capítulo define y desarrolla el modelo de programación para la creación de un DSL con gramática XML sobre un CSWE (O2) basados en el PsiEngine (O1). Se presentará una visión general (sección 4.1) y los elementos de programación que conforman el **Modelo de Programación Psi,** denotado por **PsiModel** (sección 4.2). PsiModel separa la especificación de los elementos de programación (Modelo de Especificación Psi) de su implementación (Modelo de Implementación Psi), es decir, aplica técnicas de *code-behind* (sección 4.3). Además, se resume tanto la especificación como la implementación del PsiModel mediante los lenguajes MPsi y MIPsi (sección 4.4).

4.1 Introducción

La generación de código en Ingeniería de Software es actualmente una herramienta de trabajo ampliamente utilizada. Su uso se traduce en ahorro de tiempo, eficiencia, calidad y estandarización para la creación de sistemas de información.

En sus inicios, las herramientas CASE permitían la generación automática de código a partir de modelos o diagramas. Los IDEs y la evolución de los DSLs y DSVLs han permitido simplificar la construcción de herramientas de generación de código. Por ejemplo, las tecnologías como ASP.NET (Active Server Page .NET) (Microsoft, 2014) y JSF (Java Server Face) (Oracle, 2014) son frameworks que combinan el uso de lenguajes HTML5, JavaScript y CSS, con lenguajes de servidor (C# o VisualBasic en ASP.NET; y Java con JSF) para la generación de páginas web dinámicas. Además, ASPX y JSF son lenguajes XML para la definición de la interfaz de usuario, facilitando la inclusión de controles, la navegación entre páginas y la asociación de modelos de datos. Todas las tecnologías mencionadas usan la técnica de separación de código de servidor del código de interfaz de usuario (*code-behind*), permitiendo que los diseñadores web y los programadores trabajen de forma independiente.

En estos términos el **Modelo de Programación Psi** (**PsiModel**, **P**rogrammable **S**olutions **I**nterpreter **Model**) en un CSWE, se concreta conforme a la siguiente definición:

> **PsiModel**
> Conjunto de elementos de programación basados en lenguajes Psi en un CSWE para generar código JavaScript y documentación, orientado a la creación de nuevos lenguajes Psi y componentes reutilizables JavaScript.

Los elementos de programación disponibles en PsiModel son: *variable*, *función*, *objeto*, *tipo enumerado*, *bloque de código*, *clase*, *web widget*, *componente* y *componente Psi.*

El objetivo central del PsiModel es facilitar la construcción de los lenguajes Psi conforme a la definición del capítulo anterior, pero también es facilitar la creación de clases, web widgets y componentes, usando técnicas de separación de código, en este caso, separando la implementación de la especificación de los elementos de programación del PsiModel.

En la Figura 4-1 se muestra la especificación e implementación de una función para el ModelPsi, y la generación respectiva de código JavaScript documentado. En la **Especificación XML** está definida la etiqueta **Function** que dispone del nombre (atributo *name*), la referencia a su implementación (atributo *implementation*) y la descripción de la funcionalidad (atributo *description*). En la **Implementación XML** se tiene la etiqueta **Function** y dispone de los argumentos de la función (atributo *arguments*) y el cuerpo de la función (contenido de la etiqueta). Se crea un identificador común entre las dos etiquetas (atributo *id*) como medio de vinculación para su posterior generación. Este identificador debe ser único dentro del PsiModel. En **Plantilla XML**, se muestra la plantilla de texto basada en un conjunto de *tokens* para crear la definición de una función en código JavaScript. Un *token* (elemento con corchetes {{token}}) es reemplazado por un atributo o contenido de las etiquetas de especificación o implementación. En el ejemplo de la Figura 4-1, para generar el **Código JavaScript**, los tokens {{name}} y {{description}} son reemplazados por los atributos *name* y *description*, respectivamente de la Especificación XML. Los *tokens* {{arguments}} y {{body}} son reemplazados por el atributo *arguments* y el contenido de etiqueta, respectivamente de la Implementación XML.

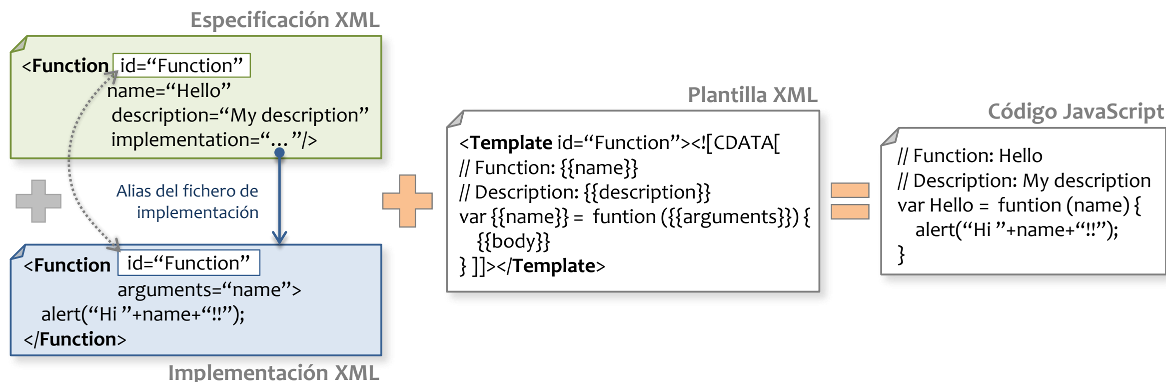


Figura 4-1. Ejemplo de especificación e implementación de una función para el Modelo de Programación Psi (ModelPsi).

El PsiModel define e implementa tanto el lenguaje de Especificación XML (lenguaje MPsi), el lenguaje de Implementación XML (lenguaje MIPsi), la Plantilla XML y la funcionalidad de generación del código JavaScript documentado, de todos sus elementos de programación. El PsiModel se basa en la programación orientada a objetos, en el lenguaje JavaScript, en la construcción de widgets y componentes y en la creación de lenguajes Psi interpretables en el PsiEngine.

Para la implementación de los lenguajes y componentes Psi de PsiModel, se seguirá explícitamente la metodología propuesta en el capítulo anterior (sección 3.4) de la siguiente forma:

P1. *Análisis y diseño de lenguajes Psi*: en esta sección se estableció la separación de la especificación de la implementación definiendo los lenguajes MPsi y MIPsi. Las secciones 4.2, 4.3 y 4.3.3 y los anexos A.1, A.2 y A.3 diseñan en detalle los lenguajes MPsi y MIPsi.

P2. *Diseño de los componentes Psi*: el diseño de los componentes MPsi y MIPsi se resume en la sección 4.4.

P3. *Implementación de los componentes Psi*: la implementación de los componentes MPsi y MIPsi se muestra en la sección 4.5.
P4. Por último, las *pruebas unitarias y funcionales* se hacen con la creación de todos los lenguajes Psi y componentes desarrollados en esta tesis, tanto los casos de estudio como los proyectos desarrollados sobre la Plataforma B2T SCOOP.

4.2 FORMALIZACIÓN DEL PSIMODEL

El primer paso de formalización es listar los elementos de programación y etiquetas de PsiModel:

EP1. **Variable** (etiqueta **Var**): define e implementa una variable.
EP2. **Función** (etiqueta **Function**): define e implementa una función.
EP3. **Objeto** (etiqueta **Object**): define e implementa un objeto.
EP4. **Tipo Enumerado** (etiqueta **Enum**): define e implementa un tipo enumerado.
EP5. **Script** (etiqueta **Script**): define un bloque de código JavaScript.
EP6. **Clase** (etiqueta **Class**): define e implementa una clase para JavaScript (ver sección 3.2.2).
EP7. **Componente** (etiqueta **Component**): define e implementa un componente JavaScript (ver sección 3.2.3).
EP8. **Web Widget** (etiqueta **Widget**): basados en la implementación Widget Factory de jQuery UI (jqueryui.com/widget).
EP9. **Componente Psi** (etiqueta **ComponentPsi**): formalizado en la sección 3.3.

Los elementos de programación Variable, Función, Tipo Enumerativo y Objeto se denominaran elementos básicos de programación del PsiModel. En resumen, el conjunto de etiquetas para los elementos de programación de PsiModel son:

$$\mathbb{T}_{EPPsi} = \left\{ \begin{matrix} \text{"Var","Function","Object","Enum","Script"} \\ \text{"Class","Widget","Component","ComponentPsi"} \end{matrix} \right\}. \quad (4\text{-}1)$$

El segundo paso de formalización, es el esquema de PsiModel para la generación de código JavaScript y documentación, como se muestra en la Figura 4-2. Este esquema está formado por el código fuente (***Fuentes de Generación Psi***) y el resultado (***Transformación Psi***). Las Fuentes de Generación Psi son la suma del *Modelo de Programación* y la *Plantilla XML*. La *Transformación Psi* es el código JavaScript generado como resultado de unir el modelo de programación con el conjunto de plantillas. Adicionalmente, el PsiModel se basa en XML-DSL y en consecuencia se pueden aplicar transformaciones XSLT para obtener documentación en formato HTML o PDF.

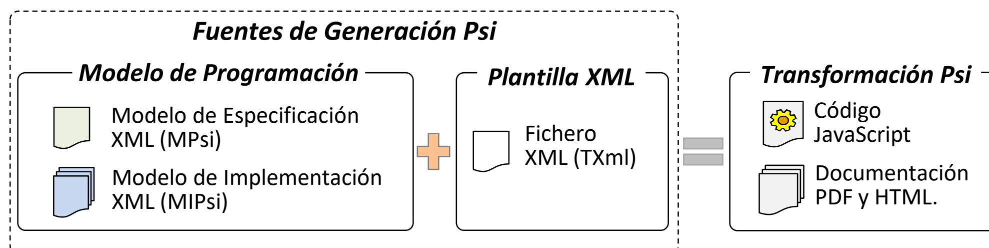


Figura 4-2. Esquema de PsiModel para la generación de código JavaScript y documentación.

El último paso de formalización es definir los lenguajes Psi y XML para PsiModel:

L1. **Modelo de Especificación Psi** (lenguaje **MPsi**): es un XML-DSL que permite especificar los elementos de programación $\mathbb{T}_{EPPsi}$.

L2. **Modelo de Implementación Psi** (lenguaje **MIPsi**): es un XML-DSL que permite implementar los elementos de programación $\mathbb{T}_{EPPsi}$, especificados mediante el lenguaje MPsi.

L3. **Plantilla XML** (lenguaje TXml): es un documento XML que escribe un conjunto de plantillas de textos con un identificador único para todos los $\mathbb{T}_{EPPsi}$.

El funcionamiento del esquema de PsiModel de la Figura 4-2, es el siguiente: mediante un proceso recursivo y el uso de las plantillas que definen todos los elementos de programación (con *Plantillas XML*), se recorre los elementos de programación en el código fuente escrito con lenguaje MPsi, y se busca su respectiva implementación en MIPsi mediante su *alias* para generar el código JavaScript. Cada *token* de la plantilla es reemplazado por un atributo de un elemento de programación, y en el caso de la implementación, por el texto del elemento.

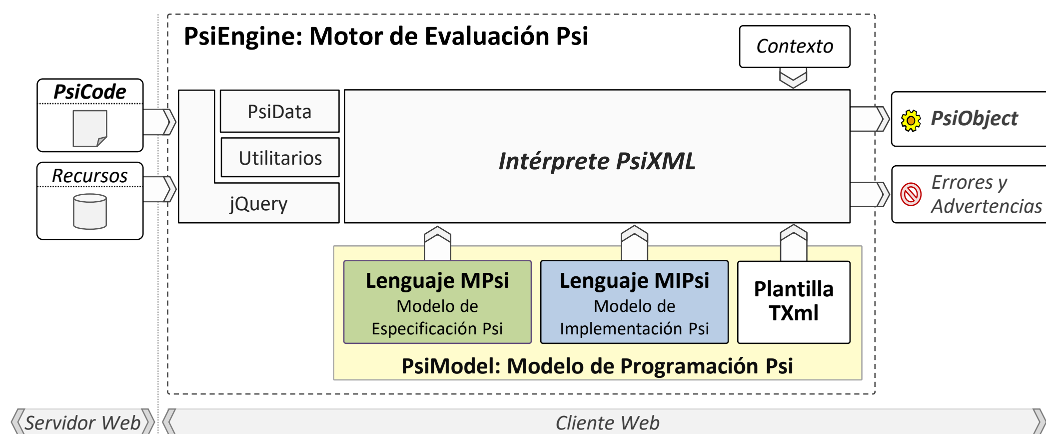


Figura 4-3. Lenguajes MPsi y MIPsi del Modelo de Programación Psi (PsiModel).

Con los lenguajes MPsi y MIPsi de PsiModel se crea el entorno de desarrollo **PsiEnvironment** para la creación de proyectos dedicados a implementar componentes y componentes Psi. El entorno PsiEnvironment se profundizará en el Capítulo 6.

Es interesante resaltar que con el PsiModel y el PsiEnvironment se han implementado todos los resultados propuestos en la tesis, tanto el PsiEngine con su PsiXML, como los lenguajes desarrollados a través de todos los capítulos, y los lenguajes y componentes desarrollados para la Plataforma B2T SCOOP.

4.3 DISEÑO DE LOS LENGUAJES PSI PARA PSIMODEL

Esta sección detalla el diseño de los lenguajes Psi y el fichero de transformación TXml mostrados en el esquema de generación de la Figura 4-2. Además se presentarán las fuentes de generación para los elementos de programación Clase, Componente y Componente Psi.

4.3.1 DEFINICIÓN DE LENGUAJES

Esta sección específica los lenguajes MPsi, MIPsi y TXml del PsiModel.

4.3.1.1 El Lenguaje MPsi

En el FRAGMENTO 4-1 se muestra la plantilla que especifica los elementos de $\mathbb{T}_{EPPsi}$ para lenguaje MPsi. El MPsi tiene como etiqueta raíz **ModelPsi**. En ella se define el nombre del modelo Psi (atributo *name*), la versión del modelo (atributo *version*), el acrónimo del modelo (atributo *acronym*) y la

descripción del modelo (atributo *description*). Este lenguaje se divide en tres partes: la *lista de ficheros de implementación*, la *definición de scripts* y la *lista de definiciones de* $\mathbb{T}_{EPPsi}$.

Los ficheros de implementación están descritos por en la etiqueta **Implementation**, donde tienen un alias de fichero (atributo *alias*) y una dirección url donde se encuentra el fichero físico de implementación (atributo *url*).

Los *scripts* son bloques de código JavaScript dentro de una etiqueta **Script**. Tiene un identificador (atributo *id*), la referencia al fichero de implementación (atributo *implementation*) y una descripción (atributo *description*).

FRAGMENTO 4-1. Plantilla para definir el Lenguaje MPsi

```
<?xml version="1.0" encoding="utf-8"?>
<ModelPsi name="…" include-header="…" version="…" acronym="…" description="…">
  <!-- Lista de ficheros de implementación -->
  <Implementation alias="…" url="…"/> …
  <!-- Lista de ficheros de script -->
  <Script id="…" description="…" implementation="…"/> …
  <!-- Definición de elementos de programación Psi -->
  <[TagEPPsi] id="…" name="…" public="…" description="…" implementation="…"> …
  </[TagEPPsi]>
  …
</ModelPsi>
```

El FRAGMENTO 4-1 [TagEPPsi] puede reemplazarse por una de las etiquetas de $\mathbb{T}_{EPPsi}$, según corresponda. La Tabla 4-1 detalla la descripción de los atributos básicos para los elementos de programación Psi.

Tabla 4-1. Atributos básicos de un elemento de programación Psi (Etiquetas $\mathbb{T}_{EPPsi}$).

Atributo	Descripción
id	Identificador del elemento de programación Psi.
name	Nombre del elemento de programación Psi.
public	Especifica si el elemento de programación en un componente es público o no.
description	Describe la funcionalidad que va a tener el elemento de programación.
implementation	Contiene el alias de referencia del fichero de implementación.

4.3.1.2 El Lenguaje MIPsi

En el FRAGMENTO 4-2 se muestra la plantilla genérica para definir la implementación de cualquier elemento de programación Psi.

La etiqueta **ModelImplementationPsi** es la etiqueta raíz del lenguaje. De forma similar a como se hace en MPsi, [TagEPPsi] es reemplaza por una etiqueta de $\mathbb{T}_{EPPsi}$, con el mismo identificador definido en su especificación (atributo *id*). Para cada elemento de programación $\mathbb{T}_{EPPsi}$ se definen atributos y etiquetas según sus requisitos. La especificación detallada junto con su definición se presentará en la sección 4.3.3.

FRAGMENTO 4-2. Plantilla para definir el Lenguaje MIPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<ModelImplementationPsi>
  <!-- Implementación de elementos de programación Psi -->
  <[TagEPPsi] id="…" …>
  …
  </[TagEPPsi]>
  …
</ModelImplementationPsi>
```

4.3.1.3 El Lenguaje TXml

En el FRAGMENTO 4-3 de define la especificación del lenguaje TXml para la Plantilla XML del PsiModel. La etiqueta raíz **TemplatesPsi** contiene un mapa de plantillas para definir los EP-Psi. Cada Plantilla (etiqueta **Template**) define un elemento de programación Psi o parte de él, y tiene una identificación única (atributo *id*).

FRAGMENTO 4-3. Definición de la Plantilla XML para el Modelo de Generación Psi.

```
<?xml version="1.0" encoding="utf-8"?>
<TemplatesPsi>
  <!— Lista de plantillas para los definir elementos de programación Psi -->
  <Template id="…">
    <![CDATA[
    // Definición de la plantilla
  ]]></Template> …
</TemplatesPsi>
```

4.3.2 EJEMPLO ILUSTRATIVO

Una vez definido los lenguajes MPsi y MIPsi y el lenguaje de transformación TXml, en la Figura 4-4 se presenta un ejemplo ilustrativo para generar una variable JavaScript documentada. El código fuente de la Figura 4-4 se explican a continuación:

- **Código MPsi** (fichero *"var.xml"*): código escrito con el lenguaje MPsi. La etiqueta **ModelPsi** es la etiqueta raíz del Modelo de Especificación Psi. Se define un fichero para su implementación (etiqueta **Implementation**) con un alias (atributo *alias*) y su URL (atributo *url*). A continuación se define una variable (etiqueta **Var**) de nombre *hello* (atributo *name*), con identificador *X* (atributo *id*), una descripción (atributo *description*) y la referencia del fichero de implementación (atributo *implementation=*"VARS").
- **Código MIPsi** (Fichero *"var-imp.xml"*): código escrito con el lenguaje MIPsi. La etiqueta **ModelImplementationPsi** es la etiqueta raíz del Modelo de Implementación Psi. En la etiqueta **Var** se escribe la implementación de la variable con identificador *X* (atributo *id*), el contenido de esta etiqueta es el valor de la variable.
- **Código TXml** (Fichero *"templates.xml"*): es la Plantilla XML. Contiene la plantilla (etiqueta **Template**) con identificador *Var* (atributo *id*) para la transformación de una variable a código JavaScript.

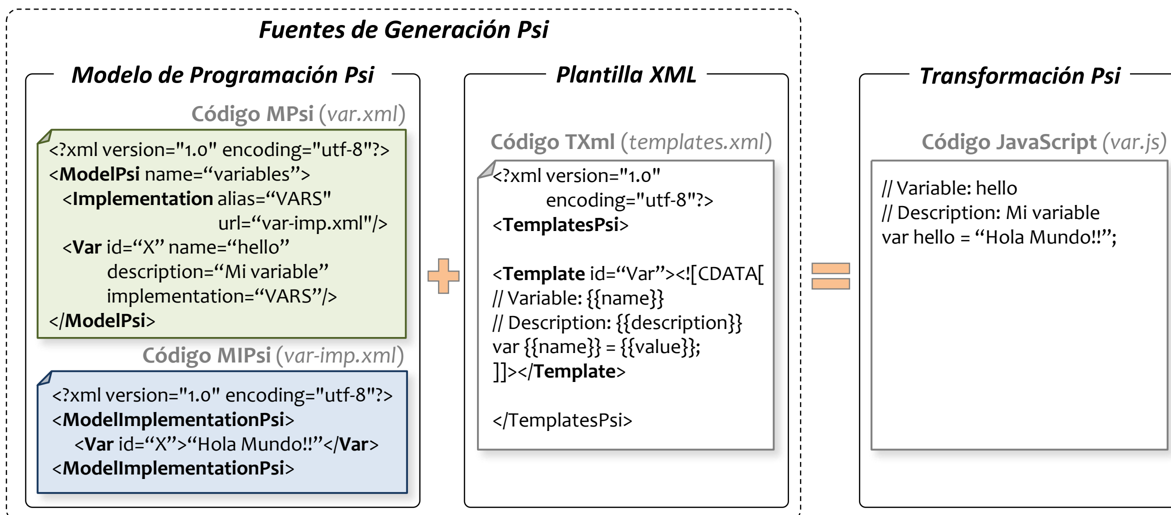


Figura 4-4. Ejemplo de fuentes de generación Psi para una variable "var" de JavaScript.

La formalización de variables y funciones presupone una complejidad adicional, pero el objetivo de usar estás etiquetas es modelar variables y funciones que tengan significado dentro de la solución del problema y que sea necesario su uso. Por ejemplo, variables globales al sistema, generación de constantes, funciones de configuración, etc. En cambio, esta complejidad en la definición simplifica la construcción de diagrama de clases y la creación de documentación.

4.3.3 Fuentes de generación Psi

Esta sección está dedicada a presentar las fuentes de generación Psi para los elementos de programación $\mathbb{T}_{EPPsi}$. Se presentaran las fuentes de generación Psi de los elementos de programación más relevantes para el PsiModel: **Class**, **Component** y **ComponentPsi**.

La fuente de generación Psi para el elemento **Var** se introdujo por medio de un ejemplo en la Figura 4-4, y para el elemento **Function** se mostró en la Figura 4-1. De igual forma, las fuentes para un tipo enumerado (elemento **Enum**) se explican en el Anexo A.1; las fuentes para un objeto (elemento **Object**) se detallan en el Anexo A.2; y las fuentes para un Web Widget (elemento **Widget**) de jQuery UI, se describen en el Anexo A.3.

4.3.3.1 Clase

En la Figura 4-5 se presentan las fuentes de generación Psi para una clase JavaScript, su etiqueta es **Class**. Además de los atributos básicos definidos en la Tabla 4-1, el atributo *inherits* define la clase padre o superclase.

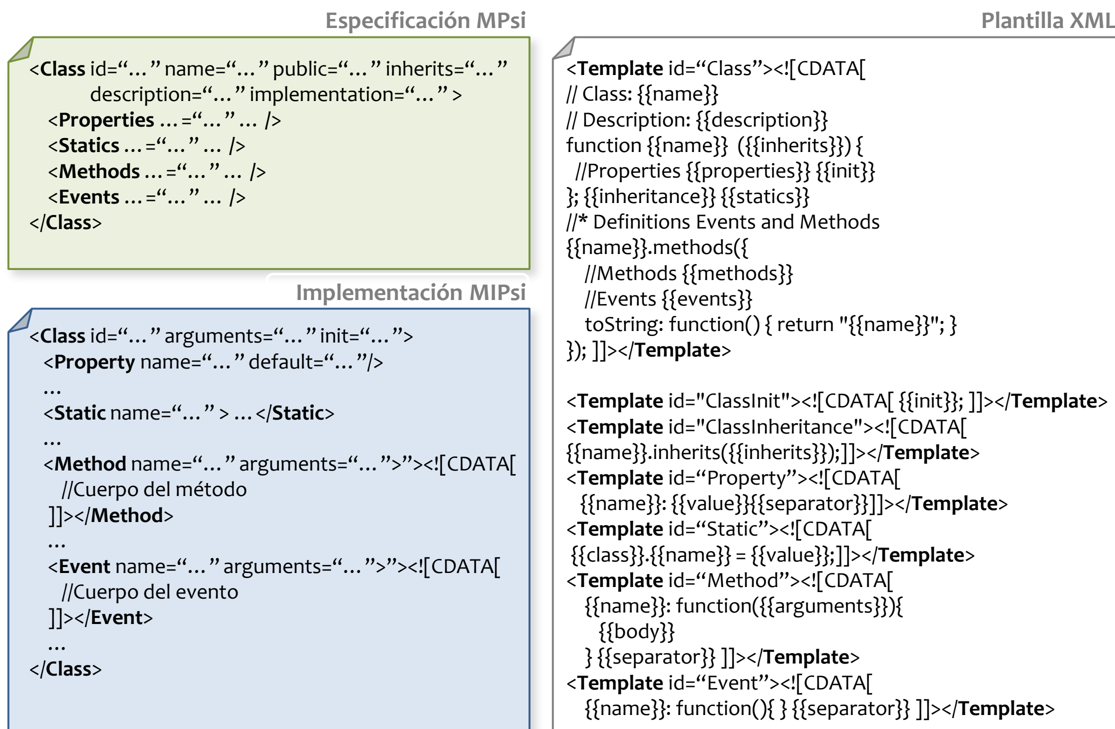


Figura 4-5. Fuentes de generación Psi para una clase "class" de JavaScript.

La Especificación MPsi de una clase está constituida por los siguientes elementos:

- Etiqueta **Properties**: lista de propiedades de la clase con sus descripciones.
- Etiqueta **Statics**: lista de propiedades/métodos estáticos de la clase con sus descripciones.
- Etiqueta **Methods**: lista de métodos de la clase con sus descripciones.

- Etiqueta **Events**: lista de eventos de la clase con sus descripciones. Aunque es la misma definición que los métodos, la diferencia aquí es conceptual, y su finalidad es poder usar programación orientada a eventos dentro de la metodología de programación Psi.

En la Implementación MIPsi de la clase, además del identificador se tienen los argumentos del constructor (atributo *arguments*) y una función de inicialización (atributo *init*). La Implementación MIPsi de una clase está constituida por los siguientes elementos:

- Etiqueta **Property**: implementa cada propiedad definida en la especificación. Una propiedad se identifica por su nombre (atributo *name*) y su valor por defecto (atributo *default*).
- Etiqueta **Static**: implementa cada propiedad/método estático de la clase.
- Etiqueta **Method**: implementa cada método de la clase. Cada método tiene definido sus argumentos (atributo *arguments*) y su cuerpo (contenido del elemento).
- Etiqueta **Event**: implementa cada evento de la clase.

En la Figura 4-5, se definen siete plantillas para generar una clase: *Class*, *ClassInherits*, *ClassInit*, *Property*, *Static*, *Method* y *Event*. Para la generación de una clase, se aplican las diversas plantillas según sea el caso: se aplica la plantilla *ClassInherits*, si el atributo *inherits* está definido y se reemplaza en {{inherits}} de la plantilla *Class*; se aplica la plantilla *ClassInit*, si el atributo *init* está definido y se reemplaza en {{init}} de la plantilla *Class*; por último, se aplican las plantillas *Property*, *Static*, *Method* y *Event* tantas veces existan, se concatenan y se cambian respectivamente en {{properties}}, {{statics}}, {{methods}} y {{events}} de la plantilla *Class*.

4.3.3.2 Componente

En esta sección se estudiará las fuentes de generación Psi para un componente (elemento **Component**) en JavaScript descrito en la sección 3.2.3. En la Figura 4-6 se muestra la Especificación MPsi y la Implementación MIPsi para un Componente, su etiqueta es **Component** y los atributos básicos se definen en la Tabla 4-1. La Especificación MPsi de un componente está constituida por:

- Etiqueta **Properties**: lista de propiedades con sus descripciones del componente, usualmente son las variables globales dentro del componente.
- Etiqueta **Methods**: lista de métodos con sus descripciones del componente.
- Etiqueta **[TagEPPsi]**: lista de elementos de programación básicos (Var, Enum, Function, Object y Class).
- Etiqueta **Component**: lista de subcomponentes. Cada subcomponente posee el mismo modelo de generación Psi que el componente, es decir, su definición es recursiva.

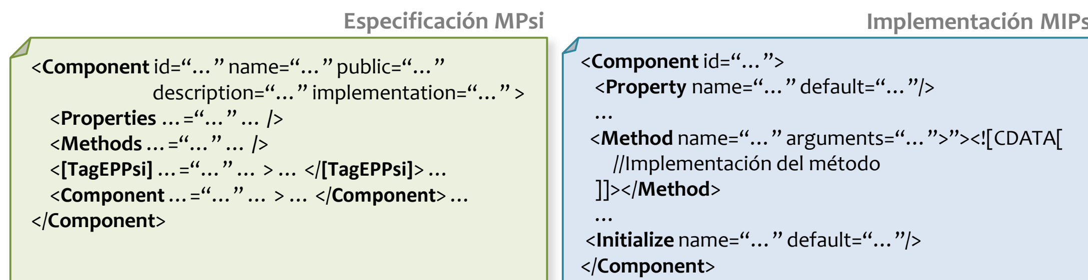


Figura 4-6. Especificación MPsi e Implementación MIPsi para un Componente.

Por su parte, la Implementación MIPsi de un componente está constituida por los siguientes elementos:

- Etiqueta **Property**: implementa cada propiedad definida en la especificación. Cada propiedad se identifica por su nombre (atributo *name*) y su valor por defecto (atributo *default*).
- Etiqueta **Method**: implementa cada método del componente. Cada método tiene definido sus argumentos (atributo *arguments*) y su cuerpo (contenido del elemento).
- Etiqueta **Initialize**: define la función de inicialización del componente.

En la Figura 4-7 se muestra la Plantilla XML para un componente. En ella se definen seis plantillas: *Component* (se basa en el FRAGMENTO 3.4), *Property*, *ComponentReturn*, *ComponentReturnContent* (contiene el esqueleto para publicar la lista de elementos de programación básicos y subcomponentes) y *ComponentInitialize* (contiene el esqueleto para la función de inicialización).

Para la generación del componente se aplican las plantillas según sea el caso:

- Aplicar las plantillas *Property* y *Method*, tantas veces existan, se concatenan y se sustituyen respectivamente {{properties}} y {{methods}} de la plantilla *Component*. Además, se aplica las plantilla *ComponentReturn* tantas veces existan propiedades y métodos, se concatenan y se reemplazan respectivamente en {{returnProperties}} y {{returnMethods}}.
- Aplicar todas las plantillas de los elementos de programación básicos y en forma recursiva el de los componentes, y se reemplaza por {{content}} de la Plantilla *Component*. Para aquellos elementos de programación básica o componentes que se han definido como públicos, se aplica la plantilla *ComponentReturnContent* tantas veces este definido, se concatena y reemplaza en {{returnContent}} de la plantilla *Component*.
- Si se define el método de inicialización, se usa la plantilla *ComponentInitialize* y se sustituye {{initialize}} de la plantilla *Component*.

Plantilla XML

```
<Template id="Component"><![CDATA[
// Component: {{name}}
// Description: {{description}}
var {{name}} = (function () {
  //Component Properties {{properties}}
  {{content}}
  //Component Methods {{methods}}
  {{initialize}}

// Component Definition: {{name}}
return {
  //Content {{returnContent}}
  //Properties {{returnProperties}}
  //Methods {{returnMethods}}
  toString: function() { return "{{name}}"; }
};
})();
]]></Template>
```

Plantilla XML (continuación)

```
<Template id="Property"><![CDATA[
  {{name}}: {{value}}{{separator}}]]></Template>

<Template id="Method"><![CDATA[
  {{name}}: function({{arguments}}){
     {{body}}
   } {{separator}} ]]></Template>

<Template id="ComponentReturn"><![CDATA[
  {{name}}: {{name}},]]></Template>

<Template id="ComponentReturnContent"><![CDATA[
  {{name}}: {{name}}, //* Type: {{type}}]]></Template>

<Template id="ComponentInitialize"><![CDATA[
// Module Initialize: {{name}}
function {{name}}() { {{body}} }
{{name}}();]]></Template>
```

Figura 4-7. Plantilla XML para un Componente.

4.3.3.3 Componentes Psi

En esta sección se presentan las fuentes de generación Psi para implementar un Componente Psi (etiqueta **ComponentPsi**). Este elemento de programación es el más relevante del ModelPsi y es la diferencia con respecto a otros lenguajes de modelado como UML, ya que implica la generación de un XML-DSL.

En la Figura 4-8 se muestra la Especificación MPsi y la Implementación MIPsi para un Componente Psi, su etiqueta es **ComponentPsi** y los atributos básicos son los definidos en la Tabla 4-1. La

Especificación MPsi de un componente está constituida por:

- La etiqueta **Properties**, la etiqueta **Methods**, las etiquetas **[TagEPPsi]** y las etiquetas **Component** son las mismas que se han definido en el componente JavaScript.
- La etiqueta **Structure** define la gramática Psi $\mathbb{G} = \langle \mathbb{T} | Root | \Delta \rangle$ de un lenguaje Psi $\mathbb{L}$. Donde $\Delta = \{Tag_i : \Delta_i | Tag_i \in \mathbb{T}\}$.

Para este último, el conjunto de etiquetas de $\mathbb{T} = \{Root, Tag_1, Tag_2, \ldots, Tag_m\}$ de un lenguaje Psi $\mathbb{L}$, es tal que, para cada etiqueta de $\mathbb{T}$, se define una etiqueta **Element** con un identificador único (atributo *key*). El identificador de la etiqueta raíz es el atributo *root* de la etiqueta **Structure**.

Para cada etiqueta **Element**, la estructura de datos Δ_i de $tag_i \in \mathbb{T}$,

$$\Delta_i = \{\text{TAG}: v_T, \text{MULTIPLICITY}: v_M, \text{CHILDREN}: v_H, \text{STRICT}: v_S, \text{VALIDATOR}: v_V\}$$

se definen como: v_T es el atributo *tag*; v_M se obtiene con el atributo *multiplicity* de las etiquetas **Child** de **Element**; v_H se obtiene con el atributo *tag* de las etiquetas **Child** de **Element**; v_S es el atributo *children-strict* de **Element**; por último, v_V se obtiene de los atributos de la etiqueta **Validator** de la clase asociada en la Implementación MIPsi.

La Implementación MIPsi de un componente Psi está constituida por los siguientes elementos:

- La etiqueta **Property**, la etiqueta **Method** y la etiqueta **Initialize** son las mismas definidas para el componente JavaScript definido anteriormente.
- Etiqueta **Validator**, como se expresó antes, sirve para crear el validador de la estructura de un lenguaje Psi $\mathbb{L} = \langle \mathbb{G} | \mathbb{K} | \mathbb{T} \leftrightarrow \mathbb{C} \rangle$. Cabe recordar que una clase de $\mathbb{C}$ puede estar asociada a diferentes etiquetas, para ello, la etiqueta **Validator** se define tantas veces sea necesario modificando su clave (atributo *element-key*) como se muestra en la Figura 4-8.
- Etiqueta **Parse** es la definición de una clase que hereda de la clase PsiXML. Parse y personaliza el comportamiento del lenguaje Psi en el momento del análisis sintáctico y semántico.

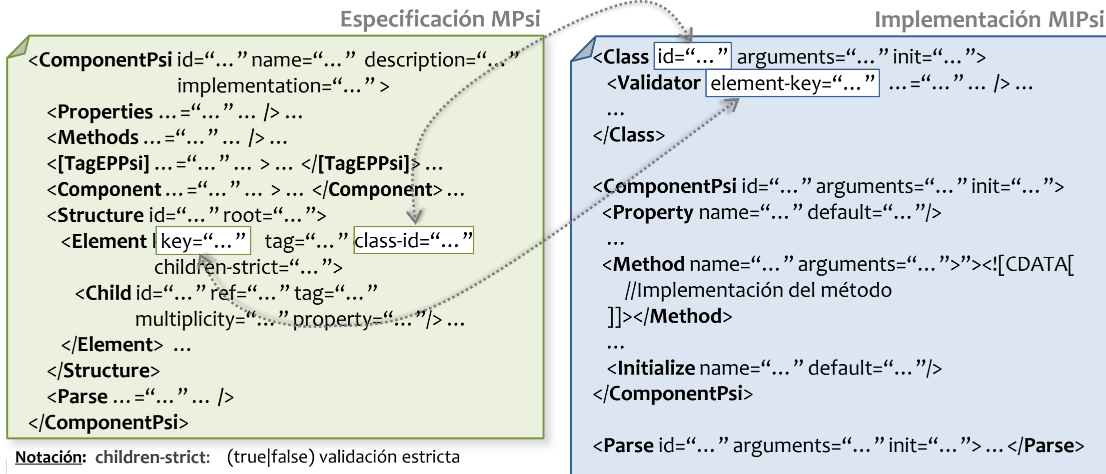


Figura 4-8. Especificación MPsi e Implementación MIPsi para un Componente Psi.

En la Figura 4-9 se muestra la Plantilla XML para un componente Psi. En ella se definen las siguientes plantillas: *ComponentPsi*, que contiene el esqueleto del componente Psi (se basa en el FRAGMENTO 3.4); *Identifiers*; *Structure*; *StructureElement* define el esqueleto de la gramática Psi (se basan en el FRAGMENTO 3.6); *Parse*; *DefineProperties*; *DefineElement*; y por último, *DefineChild* que contiene el esqueleto para el analizador del lenguaje Psi.

Para la generación del componente Psi se usan las plantillas de la siguiente forma:

- Aplicar todas las plantillas de un componente para generar el componente Psi, aplicable a la plantilla *ComponentPsi*.
- Aplicar la plantilla *Identifiers* para crear los identificadores de la gramática, concatenando el resultado y reemplazados en {{identifiers}} de la plantilla *Structure*. La plantilla *StructureElement* toma información de la etiquetas **Element** y **Child**, para crear la estructura del lenguaje y reemplazarla en {{elements}} de la plantilla *Structure*. Generada la definición del lenguaje, esta es reemplazada en {{structure}} de la plantilla *ComponentPsi*.
- Aplicar la plantilla *Parser* para personalizar la clase analizador del lenguaje Psi. Se usa las plantillas de *Class* para crear las propiedades, métodos, métodos/propiedades estáticas y sobrescribir los eventos heredados de la clase base PsiXML.Parser. La plantilla *DefineProperties* se encarga de crear el método ***defineProperties*** que es el utilizado por el análisis sintáctico y semántico para analizar cada uno de los elementos Psi, esto con la ayuda de la estructura del lenguaje Psi y las plantillas *DefineElement* (crea los {{elements}} de la plantilla *DefineProperties*) y *DefineChild* (crea los {{childs}} de la plantilla *DefineElement*). Una vez creado el método *defineProperties* este es reemplazado en {{define}} de la plantilla *Parser*. Al terminar la generación del analizador del lenguaje Psi, este es reemplazado en {{parse}} de la plantilla *ComponentPsi*.
- Aplicar la plantilla *ComponentPsi* para agregar de forma automática: el registro del lenguaje al Intérprete PsiXML, el método *run* para ejecutar un programa Psi de este nuevo lenguaje Psi y el método *language* que devuelve el apuntador a la definición del lenguaje Psi.

Plantilla XML

```
<Template id="ComponentPsi"><![CDATA[
var {{name}} = (function () {
 {{structure}}
 // Component:  Properties  {{properties}}
 {{content}}
 //Component:  Methods {{methods}}
 {{parse}}

 // Register Component to PsiXML:  {{name}}
 PsiXML.registerLanguagePsi(
    ALIAS, GRAMMAR,   {{parserClass}}
 );
 function _run(name, options, context){
    return PsiXML.traductor_PxmlToPobj(
   ALIAS, name, options, context);
 }
 function _language(){
    return PsiXML.LanguagePsi.get(ALIAS);
 }
 {{initialize}}

 //Component Definition: {{name}}
 return {
   //Content {{returnContent}}
   //Properties {{returnProperties}}
   //Methods {{returnMethods}}
   toString: function() { return "{{name}}"; },
   run: _run,
   language: _language
 };
})(); ]]></Template>
```

Plantilla XML (continuación)

```
<Template id="Identifiers"><![CDATA[
var {{nameId}} = "{{id}}";]]></Template>

<Template id="Structure"><![CDATA[
// Constants Identifiers Grammar: {{grammar}}
{{identifiers}}
ID_TAGS = {};

// Structure to GrammarPsi: {{grammar}}
{{elements}}

var GRAMMAR = {
     NAME: "{{grammar}}",
     TAGS: ID_TAGS,
     ROOT: {{root}}
   };
var ALIAS = "{{grammar}}";
]]></Template>

<Template id="StructureElement"><![CDATA[
ID_TAGS[{{nameId}}] = {
  CLASS: "{{full}}",
  TAG: "{{tag}}",
  MULTIPLICITY: {{multiplicity}},
  CHILDREN: {{children}},
  STRICT: {{strict}},
  SEND_CONTEXT: {{send}},
  VALIDATOR: {{validator}}
};
]]></Template>
```

Plantilla XML (continuación)

```
<Template id="Parser"><![CDATA[
function {{name}} ({{constructor}}) {
  //Properties {{properties}} {{init}}
};
{{name}}.inherits(PsiXML.Parser);{{statics}}
//* Definitions Events and Methods
{{name}}.methods({
  {{define}}
  //Methods {{methods}}
  //Events {{events}}
  toString: function() { return "{{name}}"; }
}); ]]></Template>

<Template id="DefineProperties"><![CDATA[
  //Define Properties
  defineProperties: function(newElement) {
    if (newElement.parent)
      switch (newElement.parent._def.key) {
         {{elements}}
      }
    },]]></Template>

<Template id="DefineElement"><![CDATA[
      case {{nameId}}:
         switch (newElement.tag) { {{childs}} };
         break;]]></Template>

<Template id="DefineChild"><![CDATA[
  case "{{tag}}":
     newElement.parent{{field}} = newElement;
break;]]></Template>
```

Figura 4-9. Plantilla XML para un Componente Psi.

4.4 DISEÑO DE LOS COMPONENTES PSI DE PSIMODEL

Esta sección se mostrará el diseño de los Compontes MPsi y MIPsi. Para cada uno, se elaborará el PsiLSD, el PsiGVA y el diagrama de clases.

4.4.1 EL COMPONENTE MPSI

El **Componente MPsi** tiene como objetivo especificar los $\mathbb{T}_{EPPsi}$. En la Figura 4-10 se muestra el PsiLSD del Lenguaje MPsi y en la Figura 4-11 el diagrama de clases del Componente MPsi.

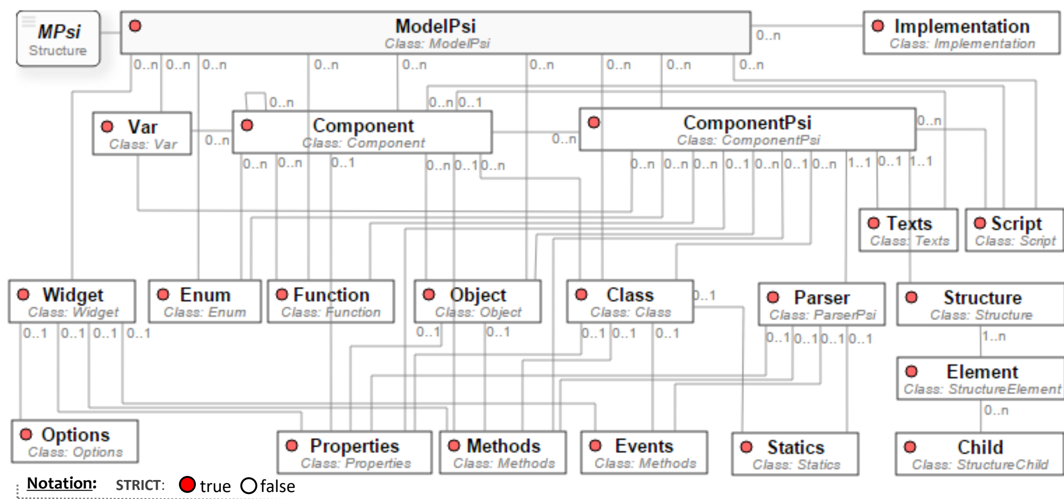


Figura 4-10. Diagrama de Estructura PsiLSD del lenguaje MPsi.

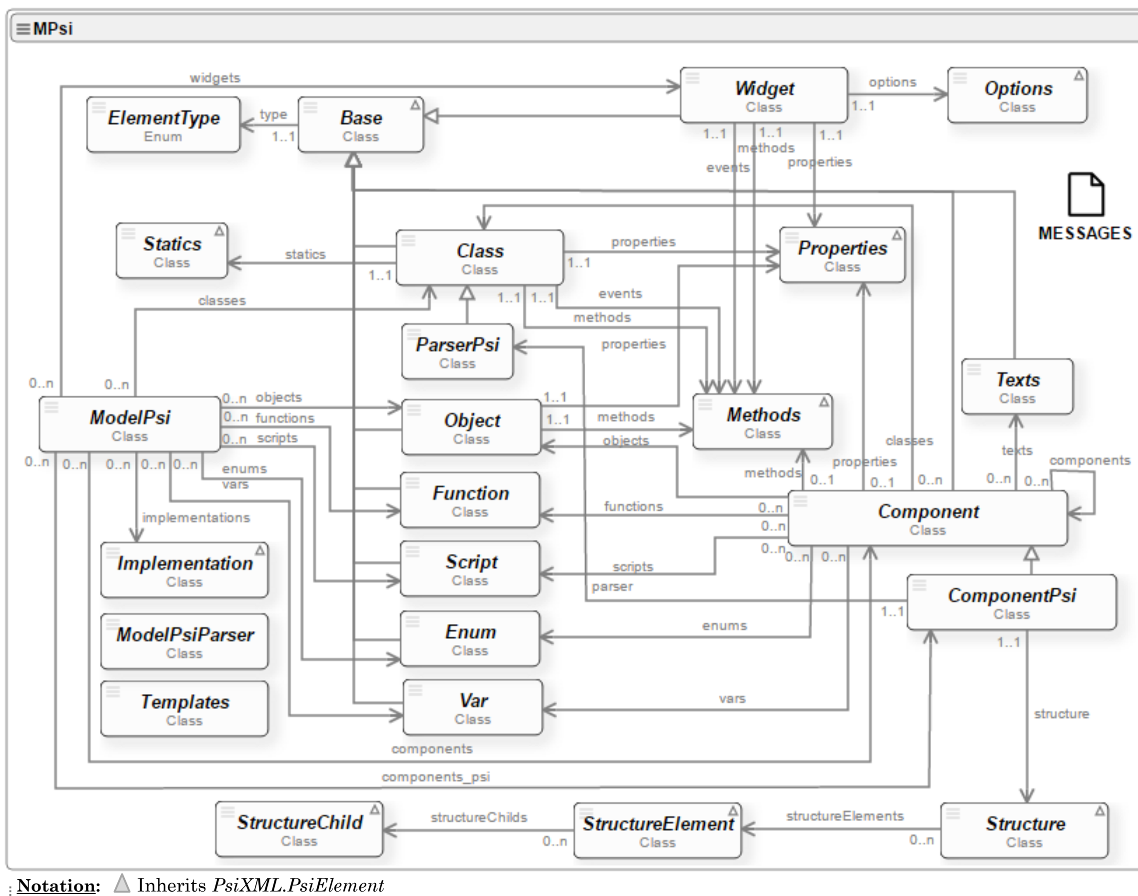


Figura 4-11. Diagrama de Clases del Componente MPsi.

En la Tabla 4-2 se encuentra una descripción de las clases asociadas para cada uno de los elementos $\mathbb{T}_{EPPsi}$.

La clase abstracta *Base* administra la referencia a la implementación para cada $\mathbb{T}_{EPPsi}$. Define tres métodos: *getInfo* (obtiene los atributos de la Tabla 4-1), *getData* (método abstracto para implementar y aplicar la plantilla de un elemento de programación) y *toJavaScript* (permite generar el código JavaScript).

El tipo enumerado *ElementType* define un objeto de identificadores para los elementos $\mathbb{T}_{EPPsi}$. La clase *Templates* administra la lista de plantillas a usar en el lenguaje MPsi, tiene una referencia al documento DOM de la Plantilla XML y construye el mapa de plantillas por nombre.

Tabla 4-2. Etiquetas y clases asociadas del lenguaje MPsi

Etiqueta	Clase	Descripción de la clase
ModelPsi* (1)	*ModelPsi**	Contiene todos los elementos de $\mathbb{T}_{EPPsi}$ (propiedades *vars*, *enums*, *functions*, *scripts*, *objects*, *classes*, *components* y *components_psi*) y la lista de implementaciones (propiedad *implementations*).
Implementation (n)	*Implementation*	Lee el atributo url y vía AJAX se obtiene el documento XML, se compila y se evalúa con el Componente MIPsi, obteniendo la referencia a un objeto de la clase ModelImplementationPsi.
Var (n)	*Var*	Define una variable y genera el código JavaScript.
Enum (n)	*Enum*	Define un tipo enumerativo y genera el código JavaScript.
Function (n)	*Function*	Define una función y genera el código JavaScript.
Script (n)	*Script*	Define un bloque de código y genera el código JavaScript.
Object (n)	*Object*	Define un objeto y genera el código JavaScript.
Class (n)	*Class*	Define una clase y genera el código JavaScript.
Widget (n)	*Widget*	Define una widget web y genera el código JavaScript.
Options (1)	*Options*	Mantiene la lista de opciones para un widget.
Component (n)	*Component*	Define una componente y genera el código JavaScript.
Properties (1)	*Properties*	Mantiene una lista de propiedades que se definen en un objeto, clase, componente y componente Psi. Genera el código JavaScript respectivo.
Methods (1), **Events** (1)	*Methods*	Mantiene una lista de métodos que se definen en un objeto, clase, widget, componente y componente Psi. Por otro lado, mantiene la lista de eventos para una clase o un widget.
Statics (1)	*Statics*	Mantiene una lista de métodos estáticos que se definen en una clase.
ComponentPsi (n)	*ComponentPsi*	Define una componente Psi y genera el código JavaScript.
Structure (1)	*Structure*	Define la estructura del lenguaje Psi.
StructureElement (n)	*StructureElement*	Define la lista de etiquetas del lenguaje Psi.
StructureChild (n)	*StructureChild*	Define la lista de referencia de las etiquetas hijas de cualquier etiqueta del lenguaje Psi.
Texts (n)	*Texts*	Administra los mensajes creados en el PsiXML y los mensajes para cualquier componente y componente Psi.

Nota: *: etiqueta o clase raíz; (1) Simple; (n) Múltiple.

4.4.2 EL COMPONENTE MIPSI

El **Componente MIPsi** específica la implementación de los elementos de programación $\mathbb{T}_{EPPsi}$. En la Figura 4-12 se muestra el PsiLSD del Lenguaje MIPsi y en la Figura 4-13 se muestra el diagrama de clases para el componente MIPsi.

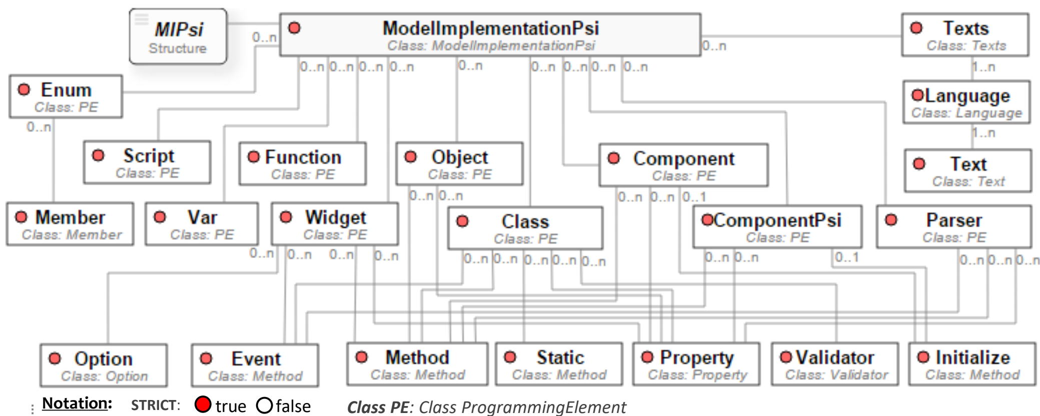


Figura 4-12. Diagrama de Estructura PsiLSD del Lenguaje MIPsi.

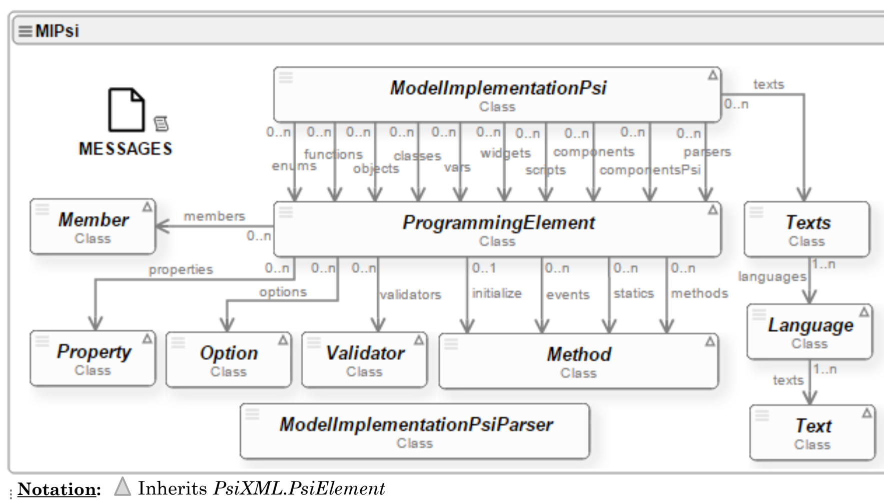


Figura 4-13. Diagrama de Clases del Componente MIPsi.

Como etiquetas adicionales, se incluyen la etiqueta **Texts**, **Language** y **Text** que manejan los mensajes de texto en diferentes idiomas de un componente o un componente Psi. En la Tabla 4-3 se detalla de las clases asociadas del lenguaje MIPsi.

Tabla 4-3. Etiquetas y clases asociadas del lenguaje MIPsi.

Etiqueta	Clase	Descripción de la clase
ModelImplementationPsi * (1)	*ModelImplementationPsi* *	Contiene todos los elementos de $\mathbb{T}_{EPPsi}$ (propiedades *vars*, *enums*, *functions*, *scripts*, *objects*, *classes*, *components* y *components_psi*) y la lista de implementaciones (propiedad *implementations*).
Texts (n), **Language** (n), **Text** (n)	*Texts*, *Language* y *Text*	Administra la información y genera los mensajes de texto en diferentes idiomas para un componente o un componente Psi.
Var (n), **Enum** (n), **Function** (n), **Script** (n), **Object** (n), **Class** (n), **Widget** (n), **Component** (n), **ComponentPsi** (n)	*ProgrammingElement*	Ésta clase se asocia a todos los $\mathbb{T}_{EPPsi}$ y con la propiedad *type* determina elemento de programación para generar la implementación del código respectivo.

Nota: *: etiqueta o clase raíz; (1) Simple; (n) Múltiple.

Tabla 4-3. (Continuación) Etiquetas y clases asociadas del lenguaje MIPsi.

Etiqueta	Clase	Descripción de la clase
Method (n), **Event** (n)	*Method*	Contiene la implementación y genera el código JavaScript de un método que se define en un objeto, clase, widget, componente y componente Psi. Igualmente para un evento de una clase o un widget.
Property (n)	*Properties*	Administra la propiedad o atributo para un objeto, clase, widget, componente o componente Psi. Genera el código JavaScript respectivo.
Option (n)	*Option*	Contiene la implementación y genera el código JavaScript una opción para el web widget.
Member (n)	*Member*	Genera el código JavaScript de un campo del tipo enumerativo.
Validator (n)	*Validator*	Administra y genera el código para el validador de atributos de la gramática Psi (PsiGVA), para un elemento de un lenguaje Psi.

Nota: *: etiqueta o clase raíz; (1) Simple; (n) Múltiple.

4.5 Implementación de los componentes Psi de PsiModel

El detalle de implementación del Modelo de Generación Psi para el PsiEngine se encuentra en la URL http://hilas.ii.uam.es/psimodel (el código fuente se encuentra en el CD adjunto y en http://github.com/echavarriaga/PsiModel). En la Figura 4-14 se muestra la página inicial para la documentación de PsiModel.

Figura 4-14. Documentación del API del Modelo de Generación Psi (PsiModel).

Para el lenguaje de Especificación MPsi, se presenta una descripción, el diagrama de clases interactivo y el API documentación, en la opción de menú "Lenguaje MPsi". De igual forma se documenta el lenguaje de Implementación MIPsi en la opción de menú "Lenguaje MIPsi".

Cabe notar que estas componentes fueron implementadas con PsiModel, es decir, se implementaron a sí mismas. El API documentación es una transformación del código fuente del proyecto MPsi. Adicionalmente, la información del diagrama de clases también fue obtenida de este mismo código

fuente y el diagrama fue construido con el modelo de programación visual que se detallará en el Capítulo 5.

4.6 Resumen

La generación de código en la Ingeniería de Software constituye, en la actualidad, una herramienta de trabajo de enorme utilidad. Está generación automática de código se traduce en ahorro de tiempo, eficiencia en programación, calidad y estandarización para la creación de sistemas de información.

En este capítulo se ha presentado el Modelo de Programación Psi (PsiModel, Programmable Solutions Interpreter Model) que implementa un conjunto de elementos de programación Psi (variable, función, script, objeto, tipo enumerativo, clase, componente, web widget y componente Psi) basados en lenguajes Psi para la generación de código JavaScript y documentación en un CSWE.

En la generación de código se han aplicado técnicas de *code-behind* que permite nseparar el Modelo de Especificación Psi (lenguaje MPsi) del Modelo de Implementación Psi (lenguaje MIPsi). Con la ayuda de la Plantilla TXml se define el proceso para generar código JavaScript (denominado Generación Psi) de los elementos de programación Psi. Asimismo, se ha siguido la metodología de creación de lenguajes Psi del capítulo anterior, para el análisis, diseño e implementación de los componentes MPsi y MIPsi.

En la Figura 4-15 se resume en un diagrama de bloques para la creación de componentes JavaScript, widgets y/o componentes Psi, basados en PsiModel.

Como se explicará en el Capítulo 6, el PsiModel es usado en el entorno de desarrollo ligero PsiEnvironment, donde se pueden definir Proyectos PsiModel. Este entorno que mencionamos facilta a los programadores simplificar el diseño e implementación de componentes JavaScript, widgets y/o componentes Psi.

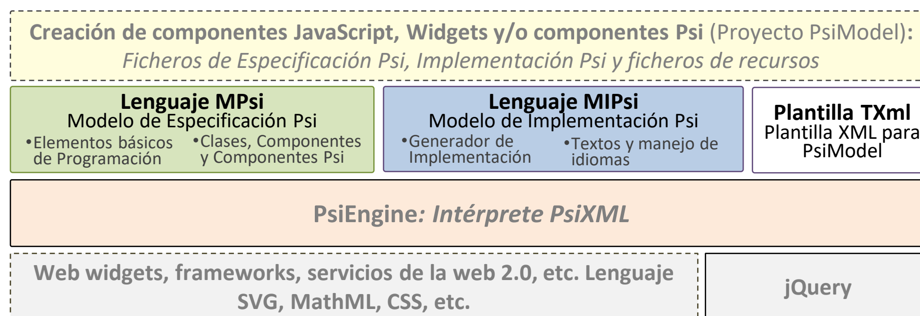


Figura 4-15. Diagrama de bloques para la creación de componentes JavaScript, widgets y/o componentes Psi, basados en PsiModel.

Capítulo 5. PSIDIAGRAM: DIAGRAMA PROGRAMABLE PSI

Este capítulo define y desarrolla el diagrama programable que facilite la creación de diagramas, la generación automática de diagramas y el diseño e implementación de DSVLs en un CSWE (O3), basados en el PsiEngine (O1). Además, se establece la generación automática de diagramas y los mecanismos de recomendación de diagramas (O4).

Para abordar este capítulo se presentará una introducción (sección 5.1), una profundización de conceptos (sección 5.2) y una formalización (sección 5.3) de diagrama programable. Tras esto, se establecerá una metodología de basada en definiciones conceptuales que facilitan el entendimiento para el diseño del diagrama programable (sección 5.4). A continuación, se presentan los conceptos de librería gráfica y diagramación (sección 5.5), herramientas visuales y utilidades de programación (sección 0), el generador de diagramas (sección 5.7) y recomendador de diagramas (sección 5.8).

5.1 INTRODUCCIÓN

Un **diagrama** es un dibujo geométrico y abstracto que representa la información de un problema, fenómeno, proposición o sistema. El concepto de diagrama en esta tesis, incluye como sinónimos modelos gráficos, lenguajes visuales, bocetos, pinturas, gráficos, etc. Por ejemplo diagramas: de estados, de bloques, de clases, de Gantt, de Venn, PERT, de procesos de negocios, de entidad-relación, etc.

Un diagrama dentro de un CSWE, se planteará como un conjunto de elementos gráficos. Cada elemento gráfico se pintará como un *gráfico vectorial bidimensional*, es decir, estará conformado por objetos geométricos independientes (líneas, arcos, polígonos, círculos, rectángulos, etc.), cada uno de ellos especificados por sus distintos atributos (radio, ancho, alto, etc.) y su forma (posición, color, grueso, etc.). Nos apoyaremos en los gráficos vectoriales redimensionables (SVG, Scalable Vector Graphics (W3C, 2011)) para definir los elementos gráficos, donde todos los navegadores lo soportan en forma nativa.

De acuerdo a lo anterior se definirá el **Diagrama Programable Psi** (PsiDiagram, **P**rogrammable **S**olutions **I**nterpreter **Diagram**) en un CSWE como:

> **PSIDIAGRAM**
>
> Conjunto de elementos gráficos que definen un diagrama basado en lenguaje SVG. Cada elemento gráfico tiene la capacidad de asociar fuentes de información heterogéneas (XML/JSON), herramientas visuales y utilidades de programación.

Atendiendo a esta definición, por un lado, las herramientas visuales disponibles en PsiDiagram serán: cajas de diálogo, barras de herramientas, ventanas emergentes, soltar, arrastrar y soltar, menús y menús flotantes. Por otro lado, las utilidades de programación serán: inclusión de ficheros y programas Psi, clases, scripts, comandos, funciones, variables, generador de diagramas y recomendador de diagramas.

En la Figura 5-1 se muestran las partes que intervienen en la creación y visualización de un diagrama programable: la **Librería Gráfica Psi** (**GLPsi**, Graphic Library Psi), las **Fuentes de Información** (**DS**, Data Source), las **Herramientas Psi** (**TPsi**, Tools Psi) y la **Diagramación Psi** (**DPsi**, Diagramming Psi).

La GLPsi tiene como objetivo definir los elementos gráficos basados en elementos de SVG que sean manipulables, transformables y reutilizables. Las DS son ficheros de datos XML o JSON asociados al diagrama y median en su apariencia y contenido. Las TPsi son las herramientas visuales y las utilidades programables asociadas al diagrama. Por último, la DPsi es la representación gráfica del diagrama con elementos SVG a partir de la GLPsi y las DS, ésta representación se hace directamente sobre un lienzo SVG (elemento **svg** de SVG) asociando las TPsi que tenga a disposición. Cabe recalcar aquí, que el generador de diagramas y el recomendador de diagramas se plantean como utilidades de programación de TPsi.

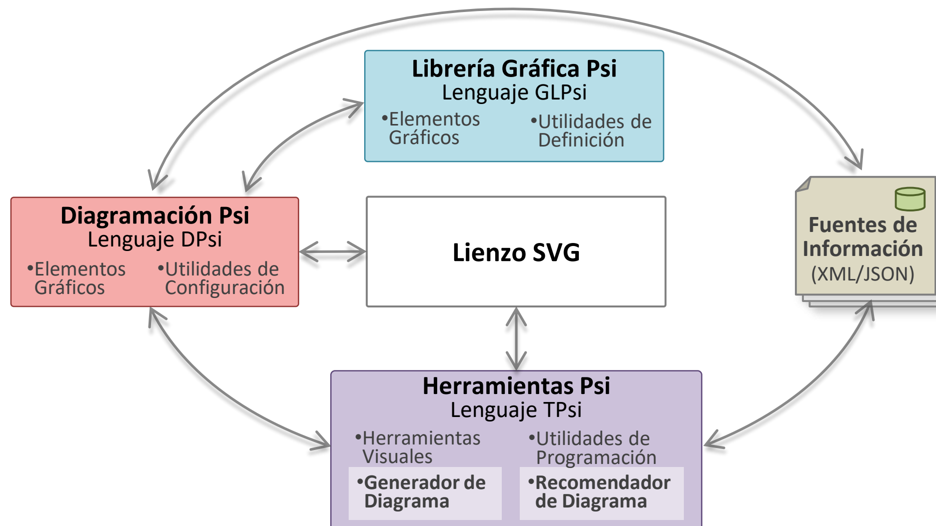


Figura 5-1. Definición de bloques de un *diagrama programable*.

Para la implementación de los lenguajes Psi de PsiDiagram, se seguirá explícitamente la metodología propuesta en la sección 3.4, se la siguiente forma:

P1. *Análisis y diseño de lenguajes Psi*: en esta sección se estableció los lenguaje Psi que median en la creación de un PsiDiagram: GLPsi, DPsi y TPsi. La sección 5.2 profundiza sobre la creación y funcionalidad de un diagrama programable y la sección 5.3 formaliza los lenguajes de PsiDiagram. La sección 5.4 establece una metodología de diseño para los lenguajes de PsiDiagram. El diseño detallado de GLPsi y DPsi se presenta en la sección 5.5.1. Por último, el análisis y diseño detallado del lenguaje TPsi se desarrollará en las secciones 5.6.1, 5.6.2, 5.7.1 y 5.8.1.

P2. *Diseño de los componentes Psi*: el diseño del componente GLPsi se muestra en la sección 5.5.2, el componente DPsi en la sección 5.5.4, y componente TPsi en las secciones 5.6.3, 5.7.2 y 5.8.2.

P3. *Implementación de los componentes Psi*: la implementación del componente GLPsi se muestra en la sección 5.5.3, el componente DPsi en la sección 5.5.5, y componente TPsi en la sección 5.9.

P4. Por último, las *pruebas unitarias y funcionales* se hacen principalmente en los proyectos desarrollados sobre la Plataforma B2T SCOOP.

5.2 CREACIÓN Y FUNCIONALIDAD DE UN DIAGRAMA PROGRAMABLE

Ésta sección está dedicada a profundizar sobre los conceptos relacionados para crear y asociar funcionalidades a un diagrama programable.

Un **elemento gráfico** se define con una plantilla de elementos SVG (rect, circle, ellipse, image, line, polyline, polygon, text, path, etc.) donde se puede especificar:

- *Elementos SVG modificables*: consiste en definir cuales elementos SVG admiten cambios de texto, cambios de color, cambios tipo de línea, cambio de color de fondo, es decir, cambios en los estilos del elemento SVG.
- *Elementos SVG de comportamiento*: consiste en definir cuales elementos SVG intervienen en el movimiento, la visualización, el tamaño, la rotación, el manejo de eventos de ratón, la asociación de herramientas visuales y funcionalidades programables.

El estudio de las diferentes herramientas de diagramación para elaborar modelos gráficos y diagramas (sección 2.4), se determinó que se puede construir un PsiDiagram a partir de los siguientes elementos gráficos (ver Figura 5-2):

E1. **Figura**: es la representación gráfica de un objeto o entidad. Se define por la agrupación de elementos SVG (etiqueta g) tales como rect, circle, ellipse, image, line, polyline, polygon, text, path, etc.

E2. **Línea:** es la descripción de un camino con un inicio y un fin, en sus extremos se definen marcadores para representar flechas, uniones, conectores, etc. También se incluye texto, tanto en sus extremos como en la mitad. Una Línea se define con un elemento line, polyline o path de SVG, mientras que sus textos con la etiqueta text de SVG.

E3. **Lista**: es la representación de un objeto que describe un conjunto de características estructuradas a través de ítems (encabezado e ítems). El encabezado y cada ítem es un conjunto de elementos SVG, respectivamente.

E4. **Contenedor**: representa un agrupador de elementos gráficos como figuras, líneas, listas u otros contenedores.

La GLPsi (Librería Gráfica Psi, parte izquierda en la Figura 5-2) tiene como objetivo definir la Línea, la Figura, la Lista y el Contenedor para que sean transformables (elementos SVG modificables), manipulables (elementos SVG modificables) y reutilizables. Como ejemplo, en la Figura 5-2 se señalan elementos SVG modificables y los elementos SVG de comportamiento.

El DPsi (Diagramación Psi, que constituye la Figura 5-2) es la representación gráfica del diagrama conformado por Líneas, Figuras, Listas y/o Contenedores, a partir de la definición de GLPsi y las DS (Fuentes de Información, parte derecha en la Figura 5-2).

Como ejemplo ilustrativo, en la Figura 5-2 el elemento *Objeto* (de Figuras) está formado por: una imagen, un texto y un círculo. Cuando se crea una instancia del elemento *Objeto* se asigna la imagen, el texto y se da un estilo de círculo (cambiar color de relleno y color de contorno). Esta modificación puede provenir de procesar la información externa al diagrama, es decir, de procesar fuentes de información XML (o JSON). En este sentido, *instanciar* un elemento gráfico es clonar todos los elementos SVG de la librería y dar una ubicación visible dentro de un lienzo SVG. Por otra parte, el elemento *Flecha* (de Líneas) está formado por: el tipo de línea y los marcadores inicial y final (obtenidos de Defs). Al igual que el elemento *Objeto*, estos elementos se pueden modificar por las fuentes de información XML/JSON. En este caso, la instancia de *Flecha* conecta dos instancias del elemento *Objeto*. Adicionalmente, en la Figura 5-2 el elemento *Panel* (de Contenedores) es un ejemplo Contenedor. Su instancia contiene a las dos instancias de *Objeto* y la instancia de *Flecha*. Por último, el elemento *Propiedades* (de Listas) tiene la

capacidad de crear una lista a partir de las fuentes de información. Para el ejemplo, toma los objetos de los “Recursos” del fichero XML, y construye la lista con el nombre y el color.

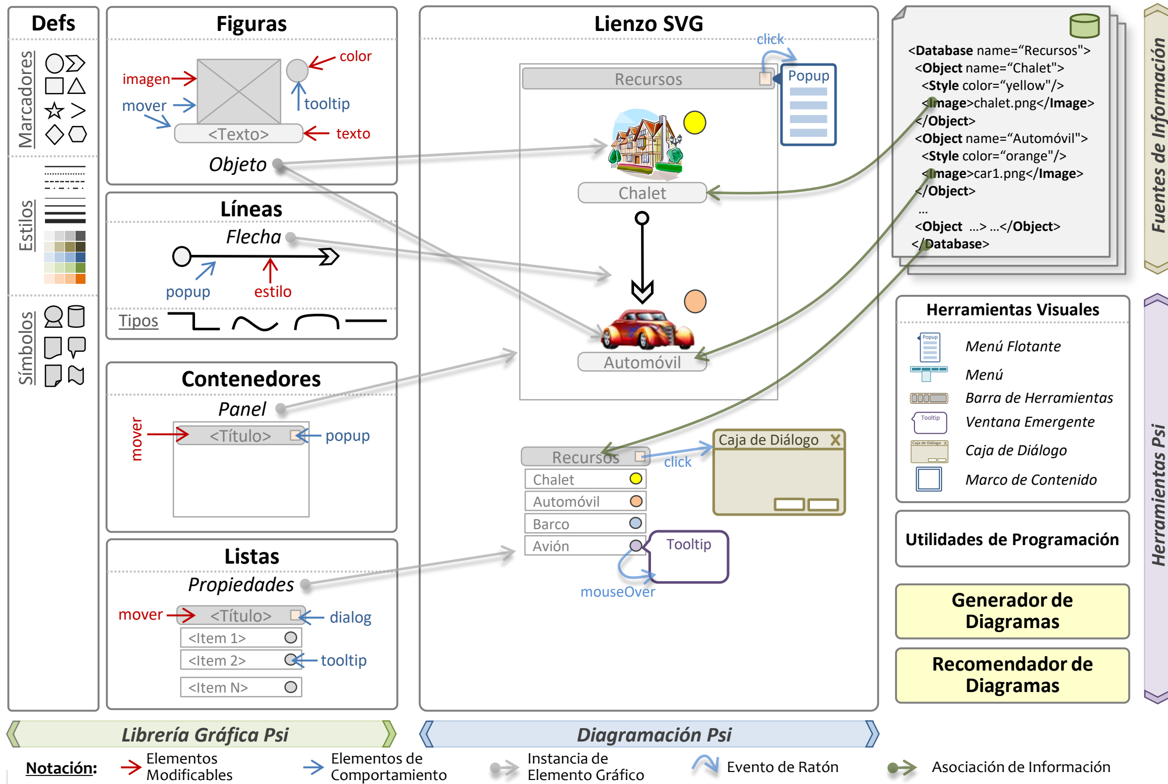


Figura 5-2. Creación y visualización de un diagrama programable. Definición de *Librería Gráfica Psi* (GLPsi), *Diagramación Psi* (DPsi), *Herramientas Psi* (TPsi) y *Fuentes de Información* (DS).

Las TPsi (Herramientas Psi, parte derecha en la Figura 5-2) la conforman las **Herramientas Visuales** (el menú flotante, la ventana emergente, la caja de diálogo, la barra de herramientas y un marco de contenido, entre otros), las **Utilidades de Programación** (clases, funciones, scripts, carga de documentos y variables), el **Generador de Diagramas** y el **Recomendador de Diagramas**. Como ejemplo ilustrativo, en la Figura 5-2, las TPsi se asocian a los elementos gráficos a través de sus elementos de comportamiento, principalmente a través de eventos de ratón de elementos SVG (evento *onclick*, *ondblclick*, *onmouseover*, etc.).

El Generador de Diagramas tiene como función crear diagramas automáticos utilizando reglas y algoritmos basados en fuentes de información heterogéneas XML/JSON en un CSWE. El Recomendador de Diagramas (O4), utiliza este generador para ofrecer diferentes opciones de visualización de diagramas al usuario, y guardar la información seleccionada para sus futuras recomendaciones.

5.3 Formalización de PsiDiagram

Resumiendo lo anterior, el objetivo de un diagrama programable, es por una parte, la *elaboración de diagramas* basados en documentos SVG donde su apariencia y contenido dependa de múltiples fuentes de información heterogéneas (XML y/o JSON). Por otra parte, se posibilita que el diagrama tenga la capacidad de *asociar herramientas y funcionalidades*, entre las que se pueden destacar las herramientas visuales (cajas de diálogo, barras de herramientas, ventanas emergentes, menús, etc.), las utilidades de

programación (clases, funciones, scripts, inclusión de ficheros, etc.), el generador de diagramas y el recomendador de diagramas.

Partiendo del diagrama general de bloques del PsiEngine, en la Figura 5-3 se muestra el **Diagrama Programable Psi**, denotado por **PsiDiagram**, encargado de crear un diagrama programable. Los componentes Psi que conforman a PsiDiagram son:

L1. **Librería Gráfica Psi** (lenguaje **GLPsi**): es un XML-DSL para definir la sintaxis y semántica concreta de un diagrama programable. Es decir, define los elementos gráficos de la **GLPsi**.

L2. **Diagramación Psi** (lenguaje **DPsi**): es un XML-DSL para pintar el diagrama sobre un lienzo SVG basado en la GLPsi, con la capacidad de asociar fuentes de información (XML/JSON) y TPsi. Es decir, implementa la representación gráfica o **diagramación** del diagrama.

L3. **Herramientas Psi** (lenguaje **TPsi**): es un XML-DSL para definir cada una de las Herramientas Visuales, las Utilidades de Programación (inclusiones, widgets, clases, funciones, variables, scripts y modificadores de elementos gráficos), el Generador de Diagramas y el Recomendador de Diagramas. Entre las Herramientas Visuales a usar se encuentran: cajas de dialogo, menús, menús flotantes, ventana emergente, barra de herramientas, marco de contenidos, soltar y soltar y arrastrar. Por otro lado, las Utilidades de Programación a implementar son: inclusiones, widgets, clases, funciones, variables, scripts y modificadores de elementos gráficos.

Los lenguajes GLPsi, DPsi y TPsi conforman el *Kernel* de PsiDiagram y son requisito indispensable para la elaboración de diagramas programables. Las secciones 5.5 y 0 están dedicadas a la implementación de estos lenguajes Psi. Cabe notar que si se requiere pintar un diagrama, sólo se utilizan los lenguajes GLPsi y DPsi, mientras que si se incluye el lenguaje TPsi se puede dotar al diagrama de herramientas visuales y funcionalidades de programación.

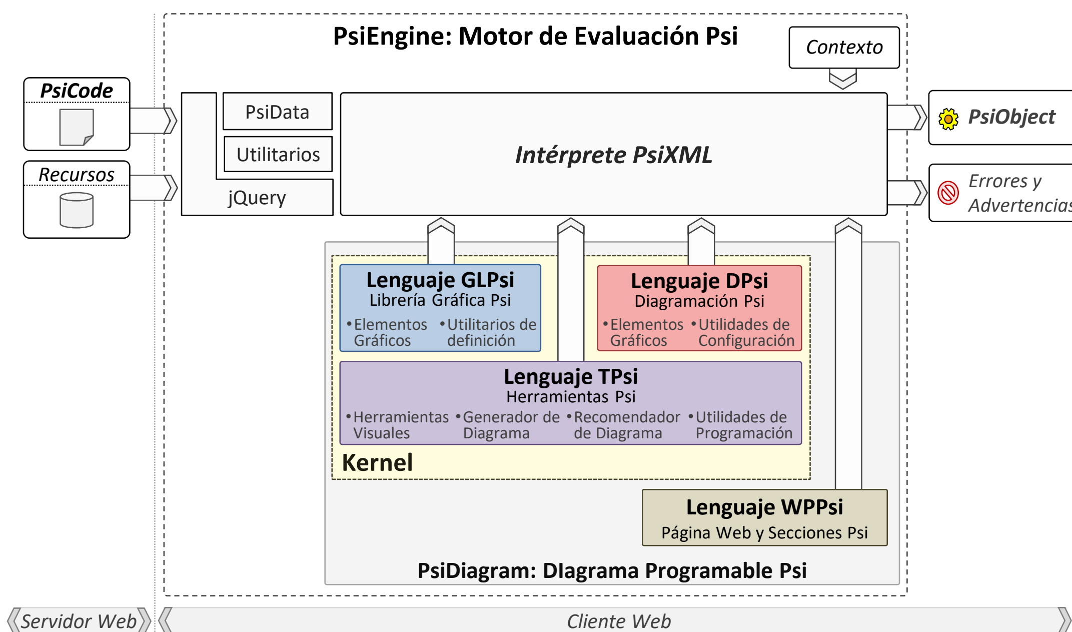


Figura 5-3. Conjunto de componentes Psi del Diagrama Programable Psi.

Adicionalmente, se creó las **Plantillas de Páginas Web Psi** (lenguaje **WPPsi**). El WPPsi es un XML-DSL para definir una página web o parte de ella, a partir plantillas definiendo secciones de distribución y llamado de contenidos. Sus contenidos pueden provenir de herramientas visuales, fragmentos de página, diagramas, marcos de contenido, etc. Esta componente no es relevante para la creación de

diagramas programables y no se presentará en este capítulo, su definición, especificación e implementación se detalla en el Anexo D.

5.4 Metodología de Diseño para PsiDiagram

Para facilitar el entendimiento y la construcción de los lenguajes GLPsi, DPsi y TPsi del PsiDiagram, en la Figura 5-4 se propone el siguiente proceso de diseño para un lenguaje Psi: el lenguaje Psi se descompone en sus principales *definiciones conceptuales* (1...N). Cada *definición conceptual* tiene una *representación* en SVG, HTML, JavaScript y/o CSS. De esta representación, se extraen los elementos necesarios y suficientes para crear la *Especificación Psi* de este concepto. La suma de todas las Especificaciones Psi permite proponer el diagrama de estructura del lenguaje (PsiLSD) y los validadores de atributos para la gramática Psi (PsiGVA). Con las definiciones, las representaciones y las especificaciones se establece el diseño del diagrama de clases del componente Psi y sus principales funcionalidades. Por último, se implementa el correspondiente componente Psi. En resumen, se aplica parte de la metodología de la sección 3.4 para la creación e implementación de lenguajes Psi a cada definición conceptual de un lenguaje Psi.

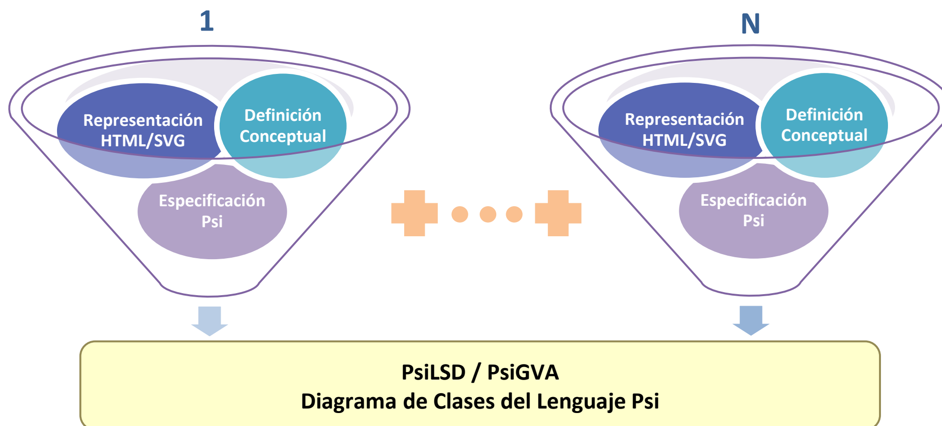


Figura 5-4. Representación del proceso de *definición conceptual*, *representación, especificación Psi* e *implementación Psi* de un lenguaje Psi.

Por ejemplo, para especificar los lenguajes GLPsi y DPsi se descompone por simplicidad en sus cuatro elementos gráficos: Línea, Figura, Lista y Contenedor. Para cada elemento gráfico se construye su definición conceptual, su representación SVG y su especificación Psi.

5.5 El Lenguaje GLPsi y el Lenguaje DPsi

En esta sección, los elementos gráficos de Línea, Figura, Lista y Contenedor se formalizarán y especificarán en el **Lenguaje GLPsi** y se pintarán con el **Lenguaje DPsi**. En consecuencia:

- Una **Línea Psi** se define como una Línea especificada en GLPsi y pintada en un diagrama DPsi.
- Una **Figura Psi** se define como una Figura especificada en GLPsi y pintada en un diagrama DPsi.
- Una **Lista Psi** se define como una Lista especificada en GLPsi y pintada en un diagrama DPsi.
- Un **Contenedor Psi** se define como un Contenedor especificado en GLPsi y pintado en un diagrama DPsi.

5.5.1 Análisis y Diseño de los Lenguajes GLPsi y DPsi

En esta subsección se presentan las definiciones y especificaciones de los elementos gráficos: Línea Psi, Figura Psi, Lista Psi y Contenedor Psi.

Para la Línea Psi se plantea en detalle la *definición conceptual*, la *representación SVG* y la *especificación GLPsi/DPsi* mostrada en el proceso de diseño de la sección 5.4. Para la Figura Psi, se resumirá su *definición conceptual* y la *representación SVG*, mientras que su *especificación GLPsi/DPsi* será adicionada como anexo. Por último, la Lista Psi y el Contenedor Psi se resumirán su *definición conceptual*, mientras que la *representación SVG* y la *especificación GLPsi/DPsi* serán adicionadas como anexo.

5.5.1.1 Línea Psi

En la Figura 5-5 se muestra la definición conceptual de Línea. Una Línea está constituida por un *punto inicial*, un *camino*, un *punto final*, *marcadores*, *rótulos* y *puntos de control*. Un ***camino*** representa la trayectoria del *punto inicial* al *punto final* y utiliza el concepto de punto actual de ubicación de una pluma, es decir, los puntos entre medias definen el arrastre de la pluma para crear concatenación de líneas rectas, líneas curvas o saltos. Por ejemplo, ir del punto A al punto B con C y D como puntos intermedios, de A a C se pinta una línea recta, desde C se dibuja un semicírculo hasta D, y seguido una línea recta hasta el punto B, como se muestra en la Figura 5-5. Los puntos intermedios de un *camino* se denominaran *puntos de control* y tienen la capacidad de moverse para modificar el *camino*.

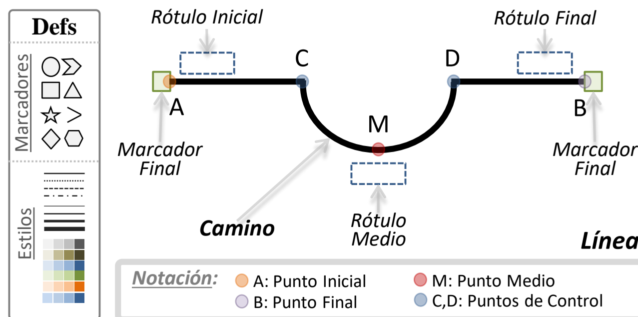


Figura 5-5. Definición conceptual de Línea Psi.

La apariencia de un *camino* se determina por su *estilo* donde, se determina el grosor, el color, la continuidad, etc. Con un *camino* se pueden crear: línea recta, línea curva, polilínea, mixta, curvas de bézier, etc. El *marcador inicial* y el *marcador final* son objetos geométricos que determinan la apariencia de los extremos (círculo, cuadrado, triángulo, etc.), creando por ejemplo, flechas, conectores, etc. Por último, los *rótulos* son textos al inicio (*rótulo inicial*), al medio (*rótulo en medio*) y al final (*rótulo final*) del camino y tienen la facilidad de moverse con respecto a su posición.

En la Figura 5-6 se muestra un ejemplo de implementación de la definición de Línea en SVG. La Línea se agrupa por el elemento **g** de SVG con un identificador (atributo *id*). El elemento **path** de SVG modela el *camino* de la Línea. El *punto inicial*, el *punto final* y los *puntos de control* están definidos en los datos de trayectoria (atributo *d*). El *punto medio* es un punto en el *camino* entre el *punto inicial* y el *punto final*, por defecto se define en el punto medio de la longitud del *camino*.

Los *marcadores* se definen como símbolos de marcado usando el elemento **marker** de SVG, este elemento es un agrupador de objetos geométricos que pinta el marcador, se especifica el ancho (atributo *markerWidth*), el alto (atributo *markerHeight*) y su punto de inflexión (atributos *refX* y *refY*) del símbolo final. Por ejemplo, en la Figura 5-6 el símbolo final es una punta de flecha con dimensiones 100x100 píxeles y su punto de inflexión es el centro, y ajustándose según la posición de la línea. El *marcador inicial* se define en el *camino* con el atributo *start-marker*, su punto de inflexión coincide con el *punto inicial*. Igualmente, el *marcador final* se define con el atributo *marker-end*, su punto de inflexión coincide con el *punto final*. Para más detalles del uso del elemento **path** de SVG se encuentra en (SVG: Paths, 2011).

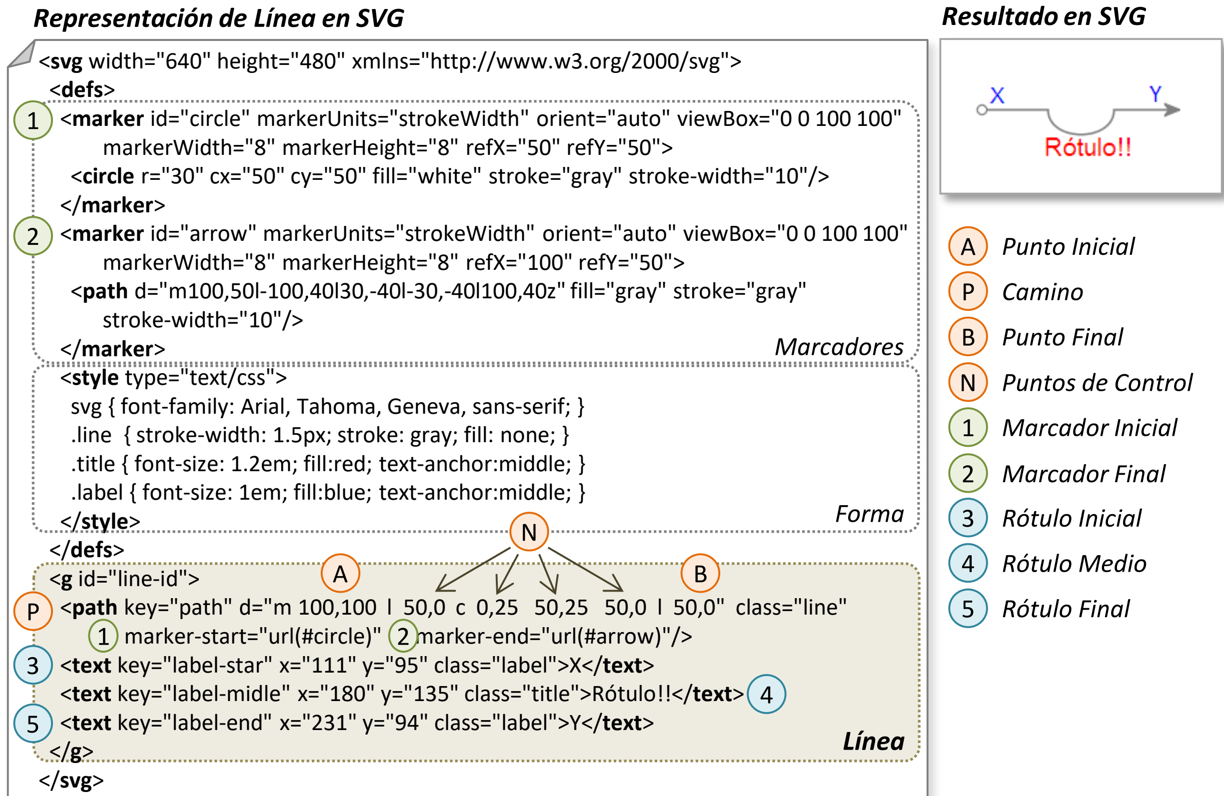


Figura 5-6. Ejemplo de implementación de la representación de Línea en SVG.

Los *rótulos* se definen con el elemento **text** de SVG. Cada *rótulo* tiene el punto de ubicación (atributos *x* e *y*) y un estilo CSS (atributo *class*). El punto de ubicación de cada *rótulo (inicial, medio* y *final)* se debe calcular como relativo a los puntos del *camino (punto inicial, punto medio* y *punto final*, respectivamente). La *forma* de una Línea se específica con lenguaje CSS en la etiqueta **style** de SVG.

Por último, en la Figura 5-7 se muestra la plantilla de Especificación GLPsi y la plantilla de Especificación DPsi para la **Línea Psi**.

- **Especificación del lenguaje GLPsi**

El GLPsi emplea la etiqueta **Defs** la cual mediante un documento SVG, escribe todas las definiciones de marcadores, estilos, símbolos, rellenos, contorno, etc., todo lo relacionado con la apariencia de un objeto geométrico. El detalle de estas definiciones se encuentra en (SVG: Painting, 2011). La Especificación GLPsi busca simplificar la definición de línea a través de la etiqueta **Line**. Tiene un identificador (atributo *id*) y el *punto medio* del *camino* (atributo *midpoint*, un número entre 0 y 1, por defecto 0.5). Y contiene dos etiquetas: **LineSetting** y **MarkSetting**.

La etiqueta **LineSetting** administra los atributos del elemento **path** SVG y define la forma (atributos *class* y *style*), la referencia al *marcador inicial* (atributo *marker-start*) y la referencia al *marcador final* (atributo *marker-end*). El resto de atributos que se definan en la etiqueta **LineSetting** son traspasados y usados por el elemento **path** de SVG, según la especificación (SVG: Paths, 2011).

El *punto inicial*, el *punto final* y los *puntos de control* se pueden visualizar con un círculo (elemento **circle** de SVG) donde la etiqueta **MarkSetting** determina su estilo (color, grueso, relleno, forma, etc.).

Estos círculos tienen la funcionalidad que al moverse dentro del diagrama, modifican el *camino* de la Línea.

Los eventos para manipular una Línea se especifican con la etiqueta **Events**. Se definen tres tipos: modificación de *puntos de control* (atributo *checkpoints*), selección de línea (atributo *select*) y arrastre de línea (atributo *drag*). Cada tipo se define con un evento de ratón sobre los elementos SVG de la Línea.

- **Especificación del lenguaje DPsi**

El DPsi contiene dos etiquetas para pintar una Línea especificada en GLPsi, la etiqueta **Line** y la etiqueta **Connector**. Su diferencia radica en que el *punto inicial* y el *punto final* de la etiqueta **Connector** se manejan mediante la definición de borde de conexión de una Figura, una Lista o un Contenedor.

Las etiquetas **Line** y **Connector** tienen un identificador único en el diagrama (atributo *id*), una clase de estilos CSS (atributo *class*) y una referencia a la definición en GLPsi (atributo *graphs*) como se muestra en la Figura 5-7. El *camino* (atributo *path*) viene definido por el siguiente micro lenguaje:

$$\langle \boldsymbol{path} \rangle \quad ::= \text{“M”} \ \langle \boldsymbol{start} \rangle \ \langle \boldsymbol{trajectory} \rangle \ \langle \boldsymbol{end} \rangle$$
$$\langle \boldsymbol{start} \rangle \quad ::= \text{“\#”}$$
$$\langle \boldsymbol{end} \rangle \quad ::= \text{“@”}$$

Donde el símbolo $\langle \boldsymbol{start} \rangle$ es el *punto inicial* sobre el diagrama y se cambia por el atributo *start*. El símbolo $\langle \boldsymbol{end} \rangle$ es el *punto final* sobre el diagrama y es reemplazado por el atributo *end*. El símbolo $\langle \boldsymbol{trajectory} \rangle$ describe el resto de los datos de la trayectoria del elemento **path** de SVG de la Línea. Para el caso de **Connector**, el *punto inicial* y *final*, es una referencia a un punto del borde de conexión.

Para un *rótulo* (*inicial, medio* y *final*) se requiere especificar el texto (atributos *start-label*, *middle-label* y *end-label*, respectivamente) y la posición relativa a su punto de origen (atributos *start-label-pos*, *middle-label-pos* y *end-label-pos*, respectivamente).

La funcionalidad de pintado de la Línea es sencilla, con la referencia a GLPsi, se solicita a GLPsi la generación del código SVG descrito en la Figura 5-7, se adicionan o modifican los atributos definidos para las etiquetas **Line** y **Connector** y se incluyen las funcionalidades descritas de los puntos de control y los eventos. Adicionalmente, las etiquetas **Line** y **Connector** contienen las etiquetas: **Setting**, **Make** y **Bind**. La etiqueta **Setting** tiene la capacidad de modificar los elementos SVG de la Línea (*camino* y *rótulos*) a

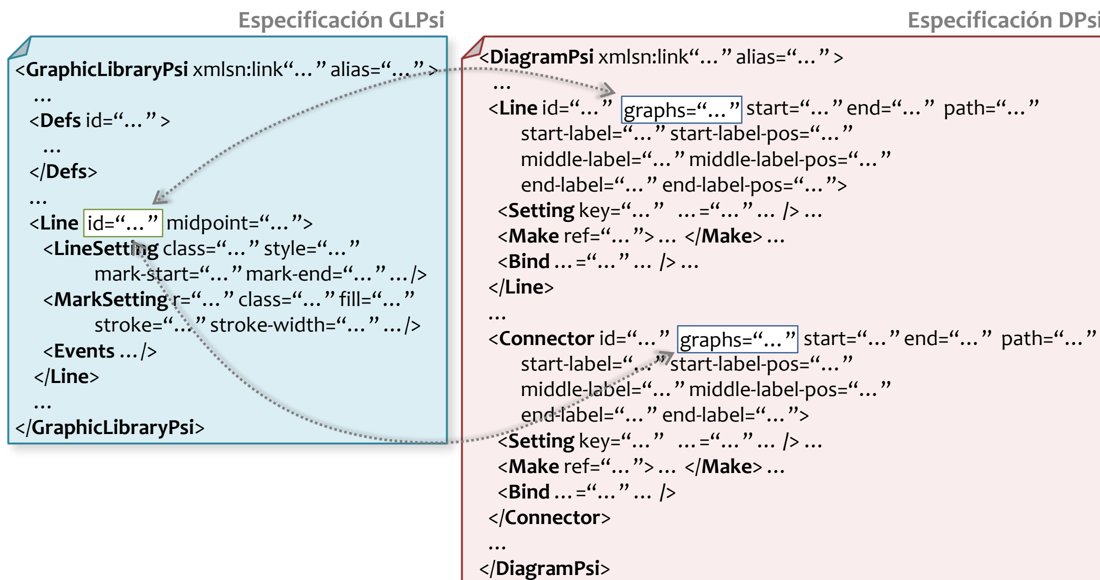


Figura 5-7. Especificación Psi de Línea Psi en el lenguaje GLPsi y en el lenguaje DPsi.

través de su clave (atributo *key*). Por ejemplo, modificar el color del *camino* o el tamaño de letra de los *rótulos*, de su definición original si se requiriese. Por otro lado, la etiqueta **Make** es una función especializada para agregar, quitar o modificar elementos SVG de la Línea. También se usa para agregar funcionalidades de movimiento, de decoración, de llamado a funciones, etc. Esta etiqueta se usa también en los otros tipos de elementos gráficos (Figura, Lista y Contenedor). Por último, la etiqueta **Bind** es la que asocia herramientas y utilidades Psi.

5.5.1.2 Figura Psi

En la Figura 5-8 se muestra la definición conceptual de **Figura**. Una Figura es la representación gráfica de un objeto, elemento o individuo. Por ejemplo, se pueden representar controles, decisiones, personas, objetos, estados, porcentajes, cajas, carpetas, documentos, bases de datos, etc., dentro de un modelo gráfico. Una Figura está constituida por *elementos básicos* y *borde de conexión*.

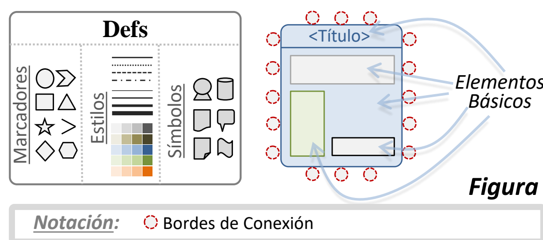


Figura 5-8. Definición conceptual de Figura.

Los *elementos básicos* son objetos geométricos como círculos, arcos, texto, líneas, imágenes, etc., a los cuales se les adiciona funcionalidades para determinar el comportamiento de la Figura. Por ejemplo, con un rectángulo de color gris y redefiniendo su evento de clic se puede: abrir una caja de diálogo, o abrir un menú, o redimensionar la figura, etc. El *borde de conexión* establece un conjunto de puntos relativos que son de la Figura y en los cuales se pueden conectar dos figuras a través de una Línea. En términos generales, el *borde de conexión* sirve para conectar figuras, listas y paneles entre sí. En la Figura 5-8 se pintan como círculos punteados para explicar en donde están ubicados, en la práctica se simulan estos puntos con círculos de colores para que se puedan ubicar dentro de la Figura y sólo se muestran cuando se están conectando dos elementos gráficos, o el conector esté cambiando de punto de conexión. Los puntos del *borde de conexión* son adaptables a cambios de tamaño de la Figura. Además, si la Figura se mueve estos puntos conservan la misma posición relativa a su origen y todos sus conectores se desplazarían a su *punto inicial*, *final* o ambos.

La apariencia de un Figura se determina por la *forma* de cada *elemento básico*. La *forma* determina el grosor, el color, la continuidad, el relleno, etc., de cada del elemento. El uso de *símbolos*, *marcadores*, *rellenos*, etc., descritos en (SVG: Painting, 2011) simplifican la construcción de la Figura, al igual que en la Lista y en el Contenedor.

En la Figura 5-9 se muestra un ejemplo de implementación de la definición de Figura en SVG. Al igual que en Línea, la Figura se agrupa por el elemento **g** de SVG con un identificador (atributo *id*). Los *elementos básicos* son definidos por:

- Figuras básicos SVG (SVG: Basic Shapes, 2011): **rect**, **circle**, **ellipse**, **line**, **polyline**, **polygon** e **image**.
- Textos SVG (SVG: Text, 2011): **text**, **tspan**, **tref** y **textPath**.
- Caminos SVG (SVG: Paths, 2011): **path**.

Los *símbolos* son muy usados en la construcción de la Figura, se definen con el elemento **simbol** de SVG, al igual que el **marker** de SVG, es un agrupador de objetos geométricos que pinta un objeto. Tiene un identificador (atributo *id*) y el marco o cuadro de visión del símbolo (atributo *viewBox*) y se referencia en el diagrama con el elemento **use** de SVG, cuantas veces sea necesario. Por ejemplo, en la Figura 5-9 el símbolo “icon” tiene un marco de visión 20x20 píxeles y contiene un rectángulo redondeado con un texto “WS”. La *forma* se define con lenguaje CSS en la etiqueta **style** de SVG. Cada elemento SVG de la Figura manipula su *forma* con el atributo *class*.

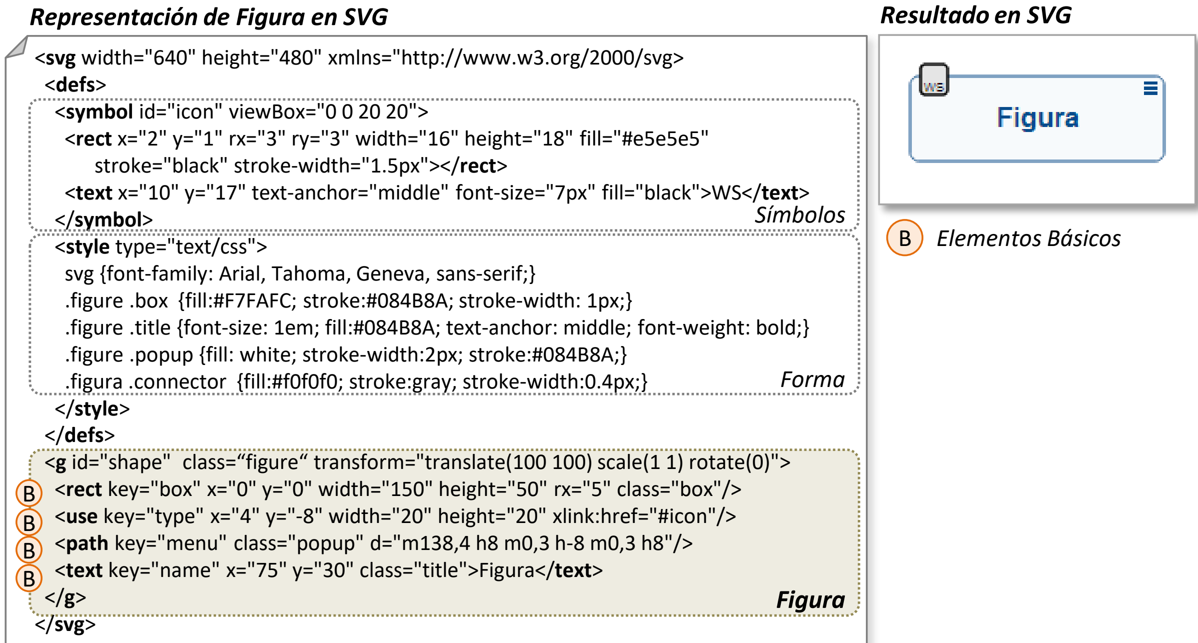


Figura 5-9. Ejemplo de implementación de la definición de Figura en SVG.

El Anexo B.1 se muestra la plantilla de Especificación GLPsi y la plantilla de Especificación DPsi para la **Figura Psi**.

5.5.1.3 Lista Psi

En la Figura 5-10 se muestra la definición conceptual de **Lista**. Cabe recordar que una Lista es la representación gráfica de un objeto que contiene una serie de características que pueden expresarse como *ítems* estructurados, y cada *ítem* contiene una información.

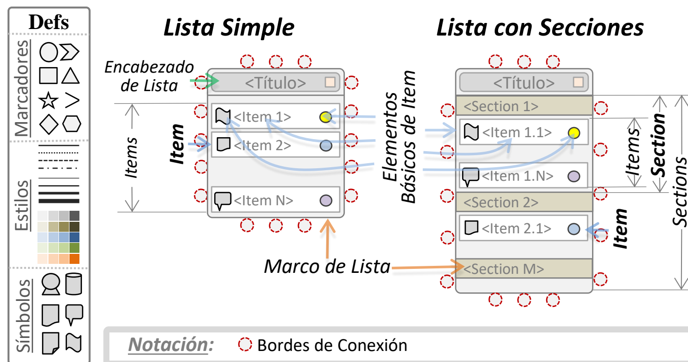


Figura 5-10. Definición conceptual de Lista.

Una Lista Simple contiene los *ítems*, el *marco*, el *encabezado* y el *borde de conexión*. Cada *ítem* y el *encabezado* son definidos con una plantilla de *elementos básicos*. Nuevamente, estos elementos son objetos geométricos que se crean dinámicamente y pueden añadir información y comportamiento. En una Lista con Secciones, los *ítems* pueden agruparse por *secciones* para dar claridad y simplicidad a la Lista. Cada *sección* es conformada por *elementos básicos*, y por lo general contienen un título y un fondo. A las *secciones* se le programan comportamientos de colapsar/expandir los ítems.

El *borde de conexión* es el descrito en la definición de Figura, y usualmente se define con respecto al *marco* de la Lista. La apariencia de una Lista se determina por la *forma* de: el *marco*, el *encabezado*, las *secciones* (si las hay) y los *ítems*. La *forma* determina el grosor, el color, la continuidad, el relleno, el texto, etc., de cada uno de los *elementos básicos* que lo componen.

En el Anexo B.2 se muestran la representación SVG, la plantilla de Especificación GLPsi y la plantilla de Especificación DPsi para la **Lista Psi**.

5.5.1.4 Contenedor Psi

En la Figura 5-11 se muestra en detalle la definición conceptual de **Contenedor**. Un Contenedor es la representación de un agrupador de elementos gráficos como figuras, líneas, listas u otros contenedores. Está constituido por: el *marco*, el *encabezado*, el *lateral derecho,* el *lateral izquierdo*, el *cuerpo*, el *pie de contenedor* y *borde de conexión*. El *encabezado* y el *pie de contenedor* son definidos con una plantilla de *elementos básicos*. Al igual que Lista, estos elementos son objetos geométricos que se crean dinámicamente y pueden añadir información y comportamiento.

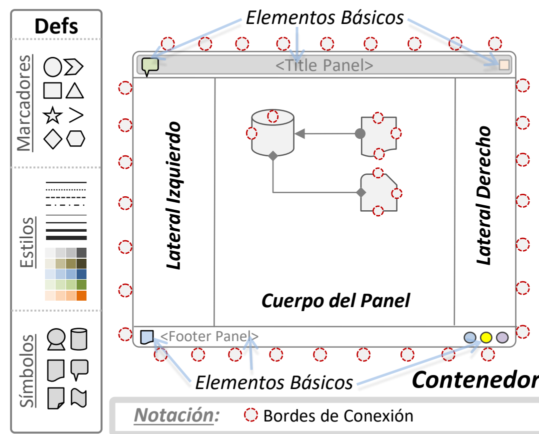


Figura 5-11. Definición conceptual de Contenedor.

Los *laterales* y el *cuerpo* del Panel contienen Líneas, Figuras, Listas y Contenedores. Es decir, es como tener un diagrama dentro de cada uno de ellos, con la ventaja de que cada elemento gráfico definido puede estar relacionado a través de conectores con otros elementos gráficos fuera del Contenedor, o pueden estar relacionados entre los *laterales* y el *cuerpo*. El *borde de conexión* es el descrito en la definición de Figura y Lista, y usualmente se define con respecto al *marco* del Contenedor.

La apariencia de un Contenedor se determina por la *forma* de: el *marco*, el *encabezado* y el *pie de contenedor*, determina el grosor, el color, la continuidad, el relleno, el texto, etc., de cada uno de los elementos básicos que lo componen.

Por último, en el Anexo B.3 se muestran la representación SVG, la plantilla de Especificación GLPsi y la plantilla de Especificación DPsi para la **Contenedor Psi**.

5.5.2 Diseño del Componente GLPsi

El **Componente GLPsi** (Graphics Library Component Psi) tiene por objetivo definir elementos gráficos basados en elementos de SVG, que sean manipulables, transformables y reutilizables.

5.5.2.1 El PsiLSD y PsiGVA de GLPsi

En la Figura 5-12 se muestra el PsiLSD del lenguaje GLPsi y en el Anexo C.1 se muestra el PsiGVA para la gramática GLPsi. La etiqueta raíz **GraphsLibrary** contiene dos conjuntos de etiquetas: *Elementos Gráficos* y *Utilidades Gráficas*.

Los *Elementos Gráficos* están conformados por las etiquetas: **Line**, **Shape**, **List** y **Container**. La etiqueta **Line** define una *Línea Psi* y cuya especificación se indicó en la sección 5.5.1.1. La etiqueta **Shape** define una *Figura Psi* y su especificación se explica en el Anexo B.1. La etiqueta **List** define una *Lista Psi* y su especificación se explica en el Anexo B.2. La etiqueta **Container** define un *Contenedor Psi* y su especificación se explica en el Anexo B.3. Adicionalmente, la etiqueta **ConnectionEdge** define los *bordes de conexión* para los elementos gráficos *Figura Psi*, *Lista Psi* y *Contenedor Psi*, y ha sido explicada su especificación en el Anexo B.1. Las etiquetas **IconSVG**, **Settings** y **Events** presentes en todos los elementos los gráficos Psi, se detallan en el Anexo B.4.

Las *Utilidades Gráficas* están conformados por las etiquetas: **Defs**, **Function**, **Script** y **Make**. La etiqueta **Defs** define símbolos, estilos CSS, marcadores, gradientes y patrones, filtros, efectos del

lenguaje SVG. Estas definiciones son usadas por los elementos gráficos. La etiqueta **Function** define una función JavaScript, con ella se manipulan y modifican los elementos gráficos. La etiqueta **Script** define un bloque de código scripts. Por último, la etiqueta **Make** sirve como modificador de comportamiento y pintado de los elementos gráficos. El detalle de la especificación de las utilidades gráficas se muestra en el Anexo B.5.

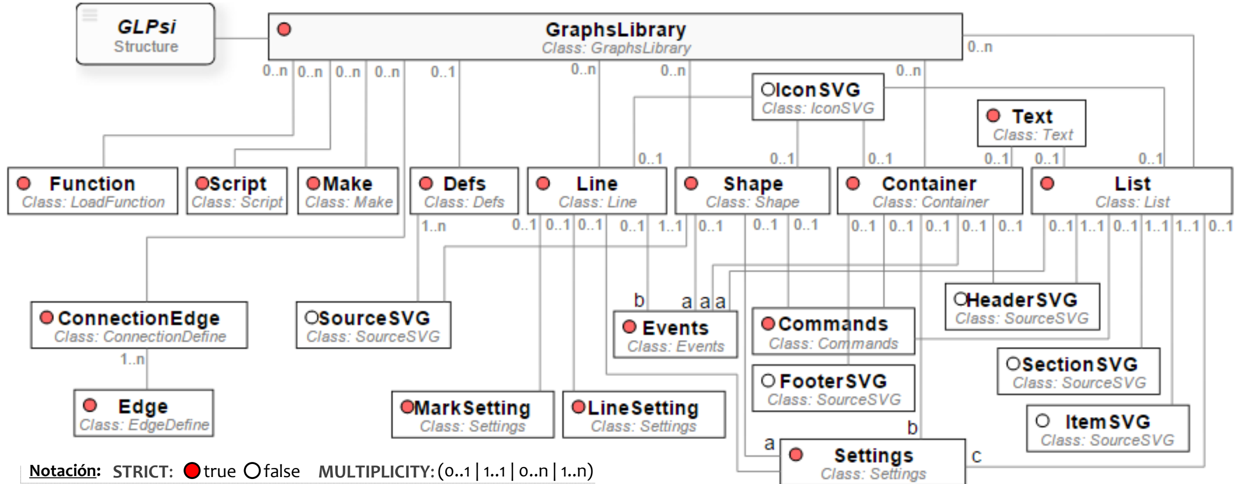


Figura 5-12. Diagrama de Estructura del Lenguaje GLPsi.

5.5.2.2 Diagrama de clases

En la Figura 5-13 se muestra el diagrama de clases simplificado del **Componente GLPsi**. El componente contiene cuatro subcomponentes: **Lines**, **Graph, Geometry** y **ConnectionEdge**.

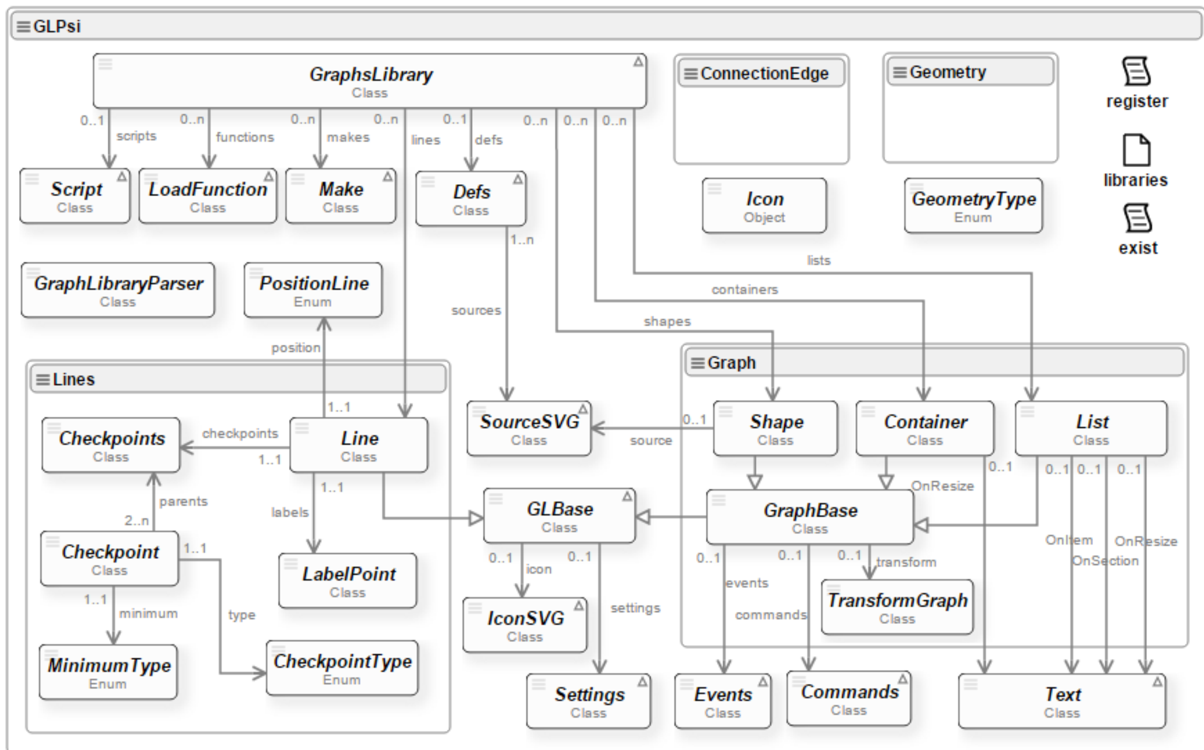


Figura 5-13. Diagrama de clases del Componente GLPsi.

El subcomponente **Lines** implementa el concepto de Línea Psi (clase *Line*). El subcomponente **Graph** implementa el concepto de Figura Psi (clase *Shape*), de Lista Psi (clase *List*) y de Contenedor Psi clase *Container*. El subcomponente **Geometry** administra la definición de punto SVG, de lista de puntos SVG y segmento de recta SVG. Por último, **ConnectionEdge** implementa el concepto de *borde de conexión*.

5.5.3 Implementación del Componente GLPsi

En la Tabla 5-1 se muestra la disposición del código fuente y la documentación para el componente GLPsi del PsiDiagram.

Tabla 5-1. Implementación y documentación del componente GLPsi

Tipo	Descripción
Componente Psi	**GLPsi** (Graphic Library Psi) http://hilas.ii.uam.es/js/psi/GLPsi.js
Documentación online	http://hilas.ii.uam.es/psidiagram (opción “Lenguaje GLPsi”)
Diagrama de clases	http://hilas.ii.uam.es/PsiXML/visor.html?url=source/models/diagram/glpsi/detail.dgl.xml&path=psi/languages/glpsi/&spec=Specification.xml
Proyecto PsiEnvironment	http://hilas.ii.uam.es/PsiXML/index.html?psimodel=GLPsi
Github	http://github.com/echavarriaga/PsiDiagram
Otros	Código fuente y documentación en CD Adjunto

5.5.4 Diseño del Componente DPsi

El **Componente DPsi** tiene como objetivo pintar un diagrama sobre un lienzo SVG basado en la GLPsi, con la capacidad de asociar fuentes de información heterogéneas y múltiples Herramientas Psi. Es decir, implementa la representación gráfica o la ***diagramación*** explicada en la sección de la visión general de PsiDiagram.

5.5.4.1 El PsiLSD y PsiGVA del lenguaje DPsi

El PsiLSD del lenguaje DPsi se dividirá en dos partes: PsiLSD para las Utilidades de Configuración (etiquetas **Params**, **Function**, **Include**, **Bind**, **Make** y **Skeleton**, ver Figura 5-14) y PsiLSD para Elementos Gráficos (etiquetas **Layer**, **Node**, **Figure**, **Connector**, **Line**, **List** y **Panel**, ver Figura 5-15). En el Anexo C.2 se muestra el **PsiGVA** para la gramática GLPsi.

En el lenguaje DPsi su etiqueta raíz es **Diagram**. Define las dimensiones y el contexto de trabajo. Esta etiqueta contiene las etiquetas de Utilidades de y de Elementos Gráficos.

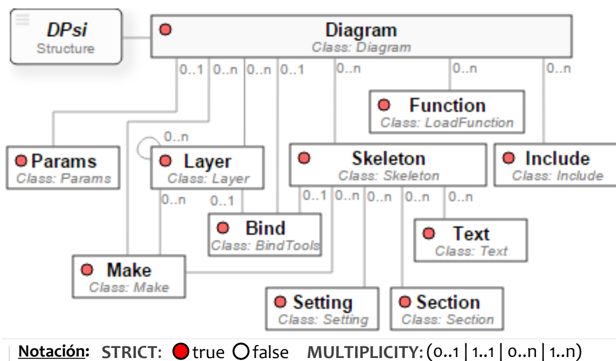


Figura 5-14. Diagrama de Estructura del Lenguaje DPsi (Utilidades de Configuación).

El detalle de la especificación de **PsiLSD para Utilidades de configuración** se encuentra en el Anexo B.6. La etiqueta **Params** define un conjunto de parámetros de entrada para la elaboración del diagrama, estos son registrados en el contexto para que cada elemento de la gramática los tenga disponible. La etiqueta **Function** define una función JavaScript, con ella se manipulan y modifican los elementos gráficos. La etiqueta **Make** sirve como modificador de comportamiento y pintado de los elementos gráficos. La etiqueta **Include** incluye un programa GLPsi o TPsi,

necesarios para el funcionamiento del diagrama programable. Por último, La etiqueta **Bind** asocia las TPsi a un elemento gráfico Psi basado en los eventos de ratón.

El detalle de la especificación de **PsiLSD para Elementos Gráficos** se encuentra en Anexo B.7. La etiqueta **Layer** define el concepto *capa gráfica* donde contiene líneas Psi, figuras Psi, listas Psi y contenedores Psi. El concepto de capa gráfica hace referencia a la profundidad en la que está el grupo de elementos de la capa, por ejemplo, si está al fondo (se define como primera etiqueta) o al frente (se define como última etiqueta).

La etiqueta **Figure** define una Figura Psi sin posibilidad de borde de conexión, mientras que la etiqueta **Node** si admite borde. De igual forma, la etiqueta **Line** define una Línea Psi, mientras que **Connector** define la misma línea pero con capacidad de conectarse a los elementos gráficos con etiquetas **Node**, **List** y **Panel**. La etiqueta **List** define una Lista Psi, donde sus secciones se definen con la etiqueta **Section** y sus ítems con la etiqueta **Item**. Por último, la etiqueta **Panel** define un Contenedor Psi, donde la etiqueta **Lateral** define los laterales derecho e izquierdo. El **Panel** contiene líneas, figuras, listas y contenedores recursivamente.

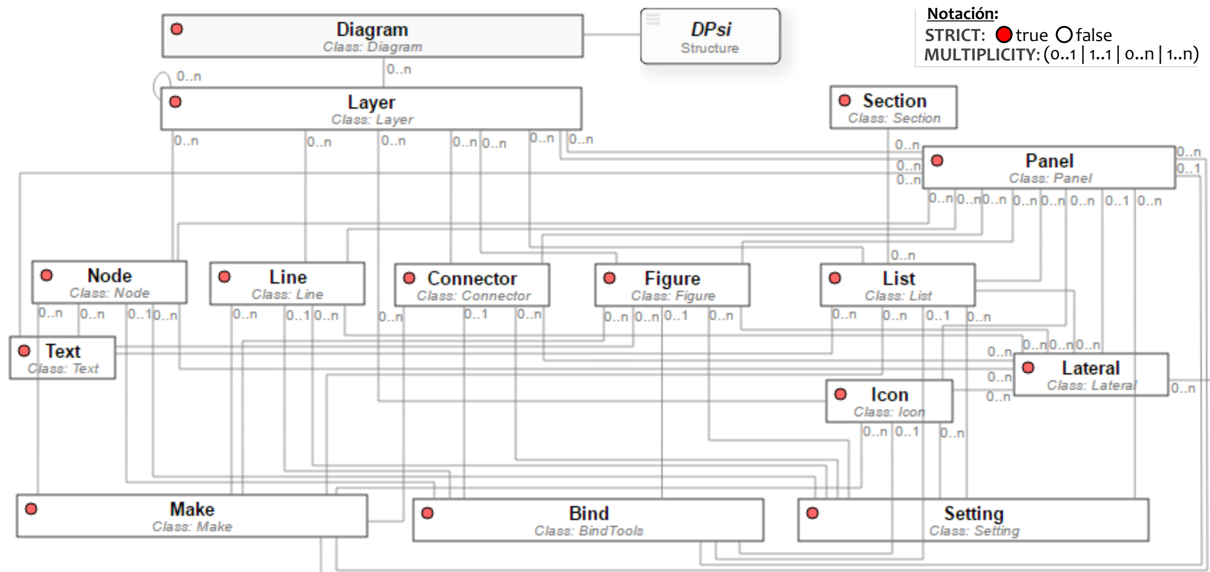


Figura 5-15. Diagrama de Estructura del Lenguaje DPsi (Elementos Gráficos).

5.5.4.2 Diagrama de clases

En la Figura 5-16 se muestra el diagrama de clases simplificado del **Componente DPsi**. El Componente DPsi implementa dos subcomponentes: **Graph** (pinta Figuras Psi, Lista Psi y Contenedor Psi) y **Lines** (pinta Línea Psi).

La clase *ManagerDiagram* administra el lienzo SVG y los elementos gráficos para un diagrama programable. Tiene la capacidad de adicionar, eliminar y buscar elementos gráficos. Además puede seleccionar elementos gráficos para el movimiento en conjunto. Puede redimensionar el lienzo SVG, modificar su escala y color de fondo. En particular, un programa DPsi se evalúa sobre una instancia de *ManagerDiagram*. De tal forma que se pueden ejecutar múltiples programas DPsi sobre el mismo diagrama potenciando las posibilidades de funcionalidad para un diagrama programable. La clase *Include* implementa la funcionalidad de cargar programas GLPsi, DPsi y TPsi. Es decir poner a disposición programas de GLPsi, TPsi y otros DPsi.

La clase abstracta *DBase* implementa las funcionalidades básicas para modificar los elementos SVG de un elemento gráfico y crear nuevas referencias. La clase *Layer* (hereda de *DBase*) implementa la

funcionalidad de *capa gráfica* y contiene líneas, conectores, paneles, nodos), figuras y otras capas gráficas. Por último, la clase abstracta *DCommon* (hereda de *DBase*) define la funcionalidad y los eventos comunes de los elementos gráficos Psi (Línea Psi, Figura Psi, Lista Psi y Contenedor Psi).

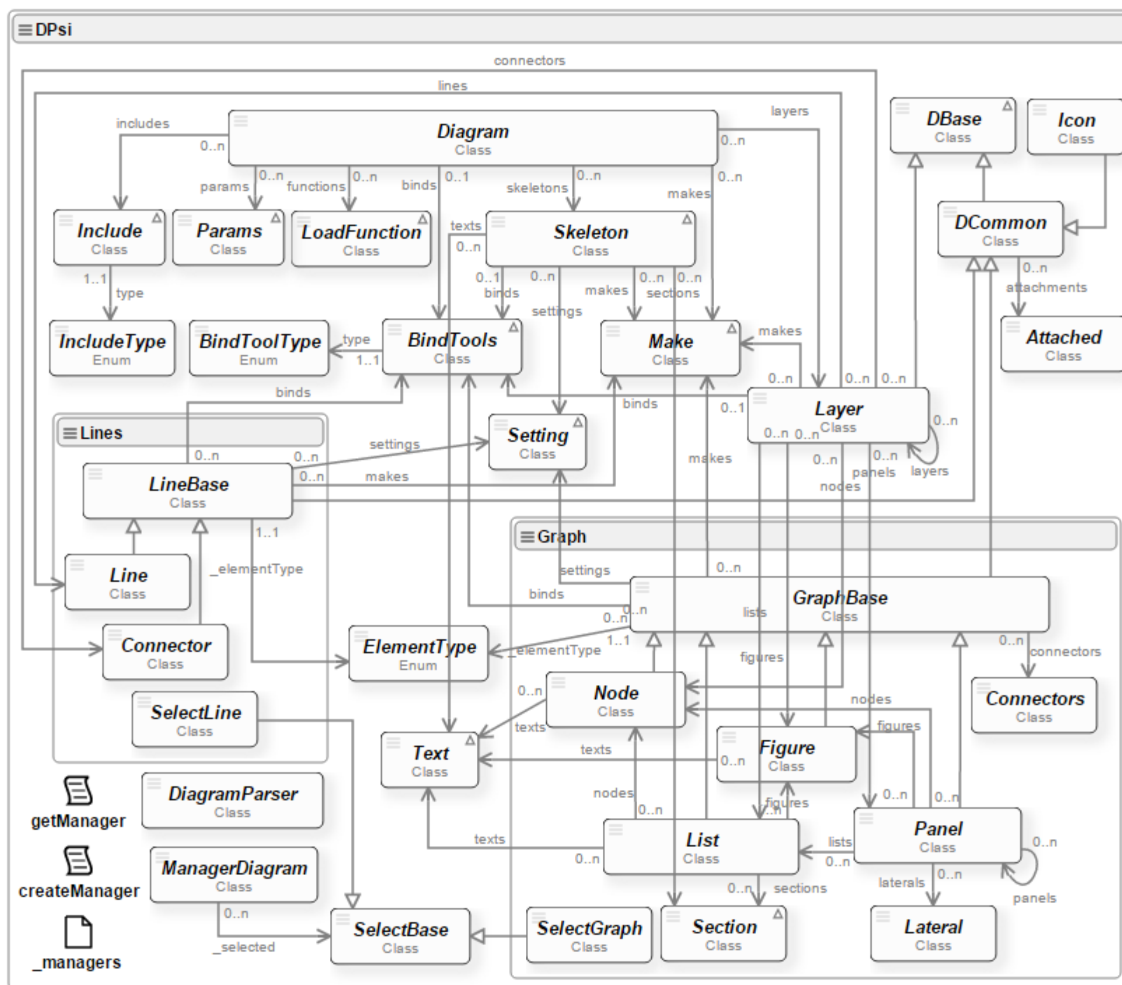


Figura 5-16. Diagrama de clases del Componente DPsi.

5.5.5 Implementación del Componente DPsi

En la Tabla 5-2 se muestra la disposición del código fuente y la documentación para el componente DPsi del PsiDiagram.

Tabla 5-2. Implementación y documentación del componente DPsi

Tipo	Descripción
Componente Psi	**DPsi** (Diagramming Psi). http://hilas.ii.uam.es/js/psi/DPsi.js
Documentación online	http://hilas.ii.uam.es/psidiagram (opción "Lenguaje DPsi")
Diagrama de clases	http://hilas.ii.uam.es/PsiXML/visor.html?url=source/models/diagram/dpsi/detail.dgl.xml&path=psi/languages/dpsi/&spec=Specification.xml
Proyecto PsiEnvironment	http://hilas.ii.uam.es/PsiXML/index.html?psimodel=DPsi
Github	http://github.com/echavarriaga/PsiDiagram
Otros	Código fuente y documentación en CD Adjunto

5.6 EL LENGUAJE TPSI

En esta sección se formalizarán, especificarán e implementarán las Herramientas Visuales (menú, menú flotante, ventana emergente, caja de diálogo, barra de herramientas, marco de contenido, soltar y arrastrar/soltar) y las Utilidades de Programación (carga de documento, clases, funciones, variables, scripts y comandos) para un *diagrama programable* en el Lenguaje TPsi. El Generador de Diagramas y el Recomendador de Diagramas se estudiarán en secciones posteriores.

5.6.1 DISEÑO DE LAS HERRAMIENTAS VISUALES

Aplicando la metodología de diseño para PsiDiagram, esta sección está dedicada a la *definición conceptual*, la *representación* y la *especificación Psi* de cada una de las Herramientas Visuales. En términos generales, una herramienta de TPsi en una página web (ver Figura 5-17) puede implementar algunos de los siguientes conceptos: *Contenido*, *Posición*, *Eventos* y *Opciones*. La especificación en Lenguaje TPsi de los anteriores conceptos se presenta en el Anexo B.8.

El *Contenido*, son los elementos visuales en código HTML, CSS, SVG, etc., que tiene la herramienta. Para crear este contenido, se pueden traer fragmentos de una página HTML, o hacer uso de información XML y transformación XSLT, o simplemente rellenar ad-doc el contenido. La *Posición* de una herramienta en una página web, es la posición (x,y) absoluta o relativa en la cual aparecerá la herramienta. Está posición se maneja usualmente con CSS sobre el Contenedor *DOM* (elemento DOM). La *posición* se especifica sobre el punto (x,y) del elemento contenedor, es decir, si está arriba, abajo, centrado, etc. Por ejemplo, en la Figura 5-17 la herramienta esta abajo y centrado del punto (x,y).

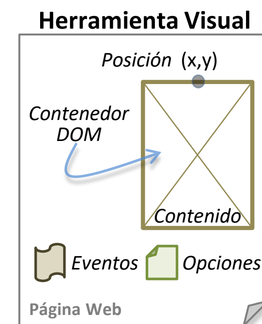


Figura 5-17. Definiciones básicas de una Herramienta Visual.

Una Herramienta Visual provee un conjunto de *Eventos* (eventos de ratón, eventos de teclado, eventos propios, etc.) para su funcionamiento, según corresponda. Adicionalmente, para las Herramientas Visuales se definen un conjunto de eventos tales como *beforeOpen* (evento antes de abrir la herramienta), *afterOpen* (evento después de abrir la herramienta), *refresh* (evento de repintar la herramienta) y *close* (evento de cerrar). Por último, las *Opciones* en una herramienta establecen la configuración para su buen funcionamiento.

5.6.1.1 Menú Flotante

En la Figura 5-18 se muestra la *definición conceptual* del Menú Flotante (etiqueta **Popup**). La *posición* coincide con la *flecha* de la ventana. Su *representación* es código HTML y define la estructura del Popup. Nuevamente, la especificación TPsi de la etiqueta **Popup** se encuentra detallada en el Anexo B.9.

La etiqueta **Popup** está conformada por las etiquetas **Item**, **Space** y **Submenu**. La etiqueta **Item** define una opción de menú y tiene la particularidad abrir una herramientas de TPsi. La etiqueta **Space** es una línea de separación de ítems y submenús. La etiqueta **Submenu** contiene subítems, separadores y submenús. Adicionalmente, un Menú Flotante puede ser abierto por un elemento gráfico de un diagrama programable, útil para asociar diferentes herramientas de TPsi a este elemento y/o editar/mostrar información asociada. También es utilizado para abrir cajas de diálogo para modificar los elementos SVG del elemento gráfico Psi.

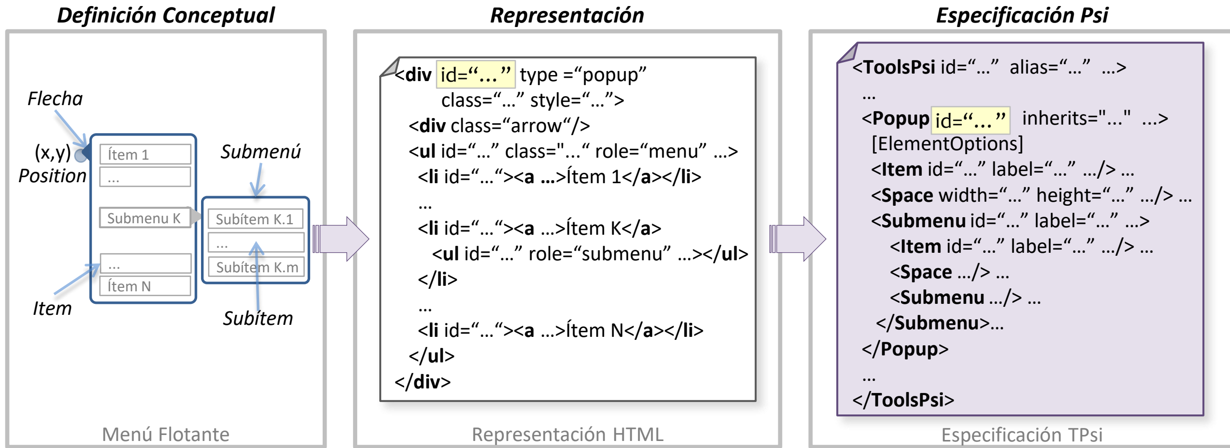


Figura 5-18. Definición conceptual, representación HTML y especificación TPsi de la herramienta Menú Flotante.

5.6.1.2 Menú

En la Figura 5-19 se muestra la *definición conceptual* de la Menú (etiqueta **Menu**). Un Menú puede especificarse en forma vertical u horizontal. Tiene la misma *representación* que el Menú Flotante pero varía en su presentación. Tiene la capacidad de abrir Herramientas Psi y su *posición* es fija. La especificación TPsi de la etiqueta **Menu** es la misma que la etiqueta **Popup** y se encuentra detallada en el Anexo B.9.

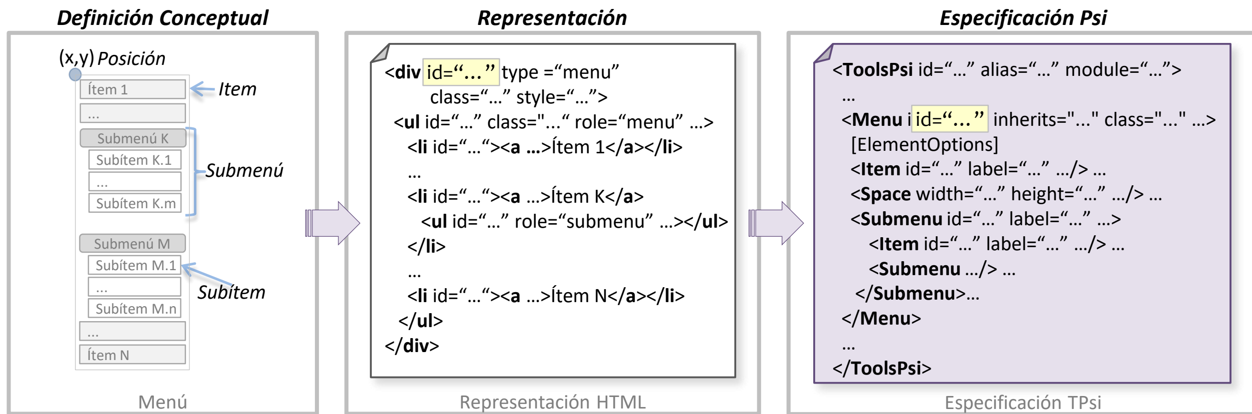


Figura 5-19. Definición conceptual, representación HTML y especificación TPsi de la herramienta Menú.

5.6.1.3 Ventana Emergente

En la Figura 5-20 se muestra la *definición conceptual* de la Ventana Emergente (etiqueta **Tooltip**). La *posición* coincide con la *flecha* de la ventana. Su *representación* es código HTML y usa CSS para su presentación. Además, en la figura se muestra el esqueleto la especificación TPsi (la especificación detallada de **Tooltip** se encuentra en el Anexo B.9). Una Ventana Emergente puede ser abierta por un elemento gráfico de un diagrama programable, útil para mostrar información asociada.

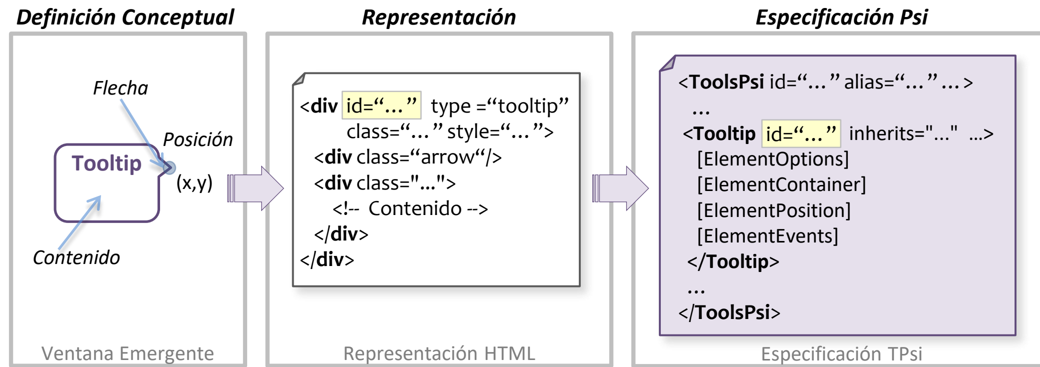


Figura 5-20. Definición conceptual, representación HTML y especificación TPsi de la herramienta Ventana Emergente.

5.6.1.4 Caja de Diálogo

En la Figura 5-21 se muestra la *definición conceptual* de la Caja de Diálogo (etiqueta **Dialog**). Al igual que la Ventana Emergente y Menú Flotante, la *posición* coincide con el punto (x,y) de la ventana. Una Caja de Diálogo tiene tres partes: *Título*, *Contenido* y *Botones*. Su *representación* es código HTML y cada parte está dentro de un contenedor DOM (elemento **div**).

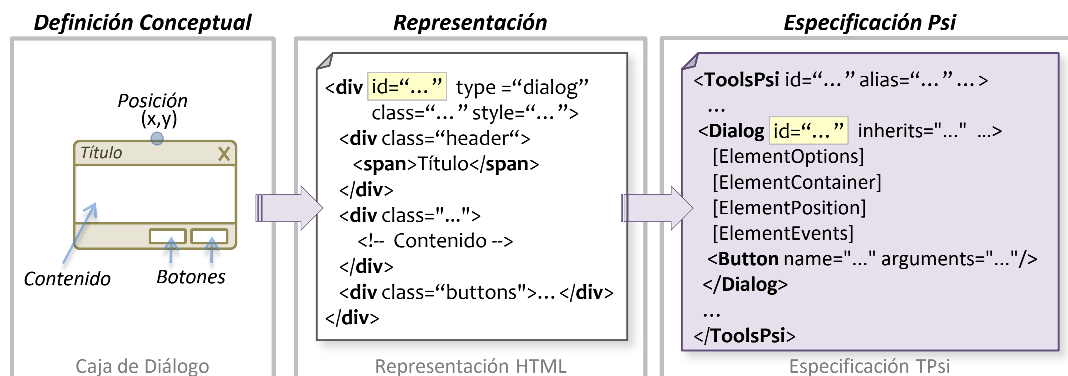


Figura 5-21. Definición conceptual, representación HTML y especificación TPsi de la herramienta Caja de Diálogo.

La especificación TPsi de la etiqueta **Dialog** se encuentra detallada en el Anexo B.9. Cabe notar que la configuración del *título*, las dimensiones, si es modal o no, etc., se encuentran en las *Opciones* de la herramienta. La etiqueta **Dialog** posee múltiples etiquetas **Button**. Cada etiqueta **Button** tiene un nombre y el código JavaScript asociado al evento *click* del botón. Nuevamente, una Caja de Diálogo puede ser abierta por un elemento gráfico de un diagrama programable, útil para configurar, asociar y modificar información relacionada con este elemento, o como herramienta de configuración de elementos gráficos.

5.6.1.5 Marco de Contenido

En la Figura 5-22 se muestra la *definición conceptual* del Marco de Contenido (etiqueta *Frame*). La *posición* es determinada por la ubicación del contenedor DOM (elemento **div**). Su objetivo es crear/traer contenidos de diversas fuentes. Por ejemplo, fragmentos HTML, transformaciones con XML, XSL y CSS, plantillas dinámicas con JSON, o cualquier widget o plugin que cree dinámicamente contenido HTML. El detalle de implementación de la etiqueta **Frame** se encuentra en el Anexo B.9.

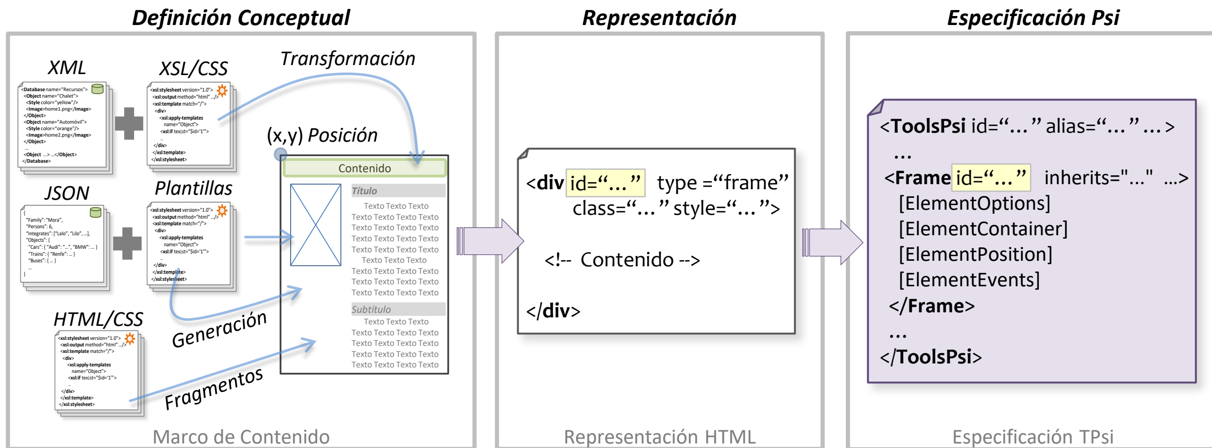


Figura 5-22. Definición conceptual, representación HTML y especificación TPsi de la herramienta Marco de Contenido.

5.6.1.6 Barra de Herramientas

En la Figura 5-23 se muestra la *definición conceptual* de la Barra de Herramientas (etiqueta **Toolbar**). Su *posición* se configura en las *Opciones*. Su *representación* es código HTML y define dos portes: el *Título* y el *Contenido*. Tiene funcionalidades como minimizar/maximizar y redimensionamiento con sus respectivos eventos.

La implementación TPsi detallada de **Toolbar** se encuentra en el Anexo B.9. En una Barra de Herramientas se configuran las herramientas visuales para un diagrama programable. También se pueden crear pequeñas widgets de monitorización y depuración.

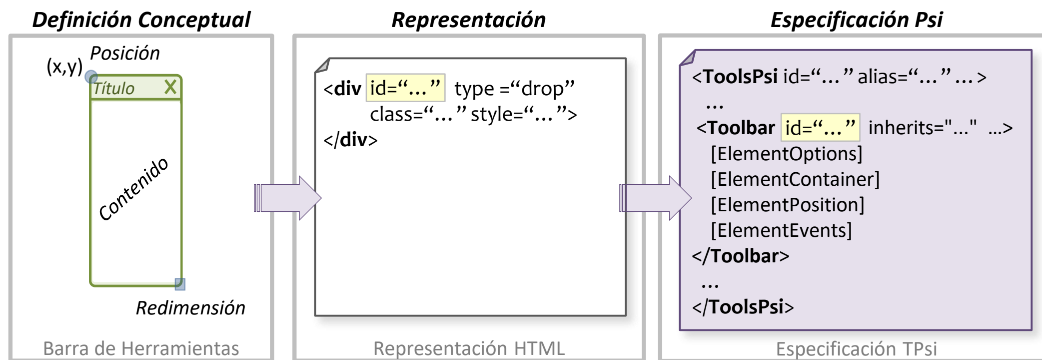


Figura 5-23. Definición conceptual, representación HTML y especificación TPsi de la herramienta Barra de Herramientas.

5.6.1.7 Soltar

En la Figura 5-24 se muestra la *definición conceptual* de la herramienta Soltar (etiqueta **Drop**). Su objetivo es proveer una *posición* (x,y) en un contenedor o lienzo, para crear contenidos o elementos gráficos. El contenedor o lienzo debe habilitar ésta capacidad de creación. Por ejemplo, en un diagrama siempre crea un elemento gráfico desde un menú, o una barra de herramientas, o una caja de diálogo. El detalle de implementación TPsi de la etiqueta **Drop** se encuentra en el Anexo B.9.

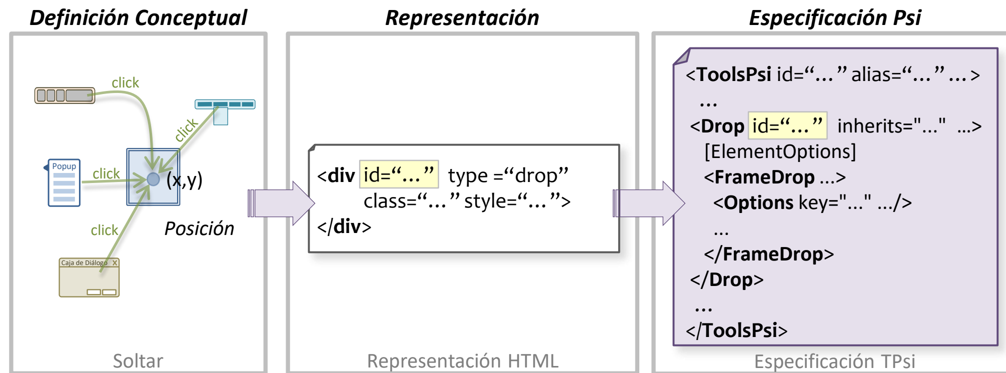


Figura 5-24. Definición conceptual, representación HTML y especificación TPsi de la herramienta Soltar.

5.6.1.8 Arrastrar/Soltar

Por último, en la Figura 5-25 se muestra la *definición conceptual* de Arrastrar/Soltar (etiqueta **DragDrop**). De forma abstracta, tiene como objetivo establecer el vínculo entre dos elementos. Para un diagrama, por ejemplo, cuando se tiene un conjunto de elementos gráficos (Figura, Lista o Contenedor), se puede activar el funcionamiento el funcionamiento de arrastrar y soltar sobre otro elemento gráfico y establecer la conexión mediante una Línea. El detalle de implementación TPsi de la etiqueta **DragDrop** se encuentra en el Anexo B.9.

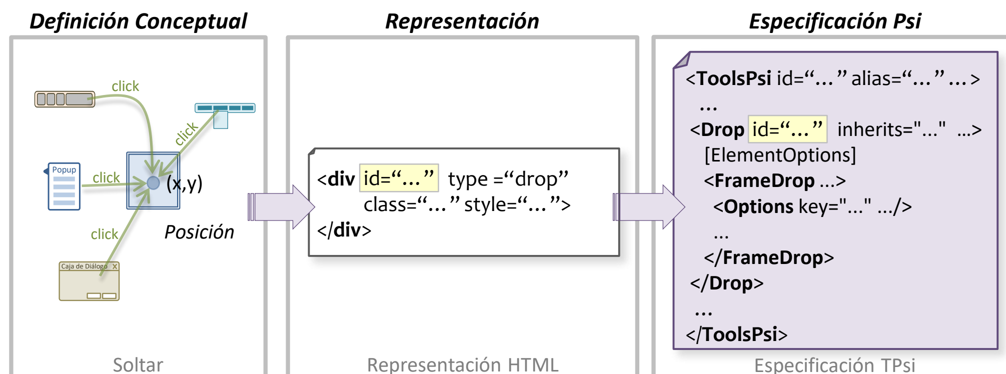


Figura 5-25. Definición conceptual, representación HTML y especificación TPsi de la herramienta Arrastras/Soltar.

5.6.2 Diseño de las Utilidades de Programación

Haciendo uso de la metodología de diseño para PsiDiagram, esta sección se dedica a la *definición conceptual*, la *representación* y la *especificación Psi* de los Utilidades de Programación para TPsi: Carga de Documento (etiqueta **LoadDocument**), Clases (etiqueta **Class**), Funciones (etiqueta **Function**), Variables (etiqueta **Var**), Comando (etiqueta **Command**), Script (etiqueta **Script**) y Modificador de Elemento Gráfico (etiqueta **Make**). La especificación Psi detallada de estas Utilidades de Programación se encuentra en el Anexo B.10.

Las Clases, Funciones, Variables y Scripts tienen como objetivo proveer de herramientas de programación orientada a objetos, basado en lenguaje JavaScript. Su especificación Psi es parecida a los elementos básicos de programación de PsiModel de las secciones 4.3 y 4.3.2. Aquí, estos elementos no separan la definición de su implementación.

El Comando es una función cuya finalidad es encadenar TPsi con un DPsi o con otras TPsi. El Comando cuando es llamado, ya sea por una herramienta de TPsi o un elemento gráfico Psi, tiene como argumentos la referencia del objeto solicitador y su *Contexto* de ejecución. La etiqueta **Command** tiene definido el cuerpo de la función (etiqueta **Body**) y las *Opciones* de la herramientas.

Por último, la Carga de Documento es una funcionalidad para traer documentos XML, JSON, XSL, etc., y ubicarlos en el PsiData del PsiEngine. La etiqueta **LoadDocument** tiene como atributos la *url* y el tipo de documento. Por último, el Script y el Modificador de Elementos Gráficos tienen la misma definición y funcionalidad que en la Diagramación Psi.

5.6.3 Diseño del Componente TPsi

El **Componente TPsi** tiene como objetivo implementar las Herramientas Visuales, las Utilidades de Programación, el Generador de Diagramas y el Recomendador de Diagramas de las **Herramientas Psi**. La implementación del Generador y Recomendador de Diagramas se mostrará en secciones posteriores.

5.6.3.1 El PsiLSD y PsiGVA del lenguaje TPsi

El PsiLSD del lenguaje TPsi se dividirá en cuatro partes: PsiLSD Herramientas Visuales, PsiLSD Utilidades de Programación, PsiLSD Generador de Diagramas y PsiLSD Recomendador de Diagramas. Adicionalmente, el PsiGVA se muestra en el Anexo C.3.

En la Figura 5-26 se muestra el PsiLSD Herramientas Visuales. Las etiquetas que lo conforman son: **Menu** (especifica el Menú), **Popup** (especifica el Menú Flotante), **Toolbar** (especifica la Barra de Herramientas), **Dialog** (especifica la Caja de Diálogo), **Frame** (especifica el Marco de Contenido), **Drop** (especifica el Soltar) y **DragDrop** (especifica el Arrastrar/Soltar). Cada una de ella se ha mencionado en la sección anterior y el detalle de su especificación se encuentra en el Anexo B.9.

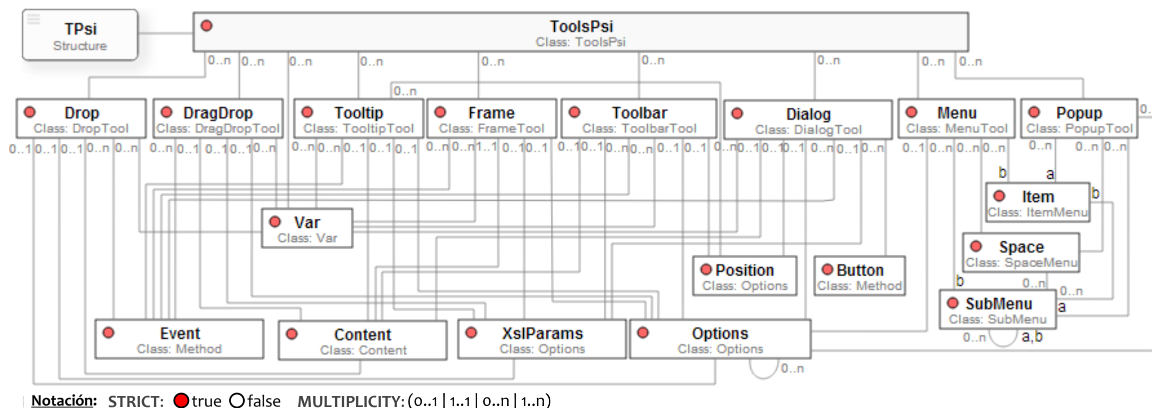


Figura 5-26. Diagrama de Estructura del Herramientas Visuales de TPsi.

De igual forma, en la Figura 5-27 se muestra el PsiLSD Utilidades de Programación. Las etiquetas que lo conforman son: **Var** (especifica una variable), **Function** (especifica una función), **Script** (especifica un script en lenguaje JavaScript), **Class** (especifica una clase), **Make** (especifica un modificador), **LoadDocument** (especifica una carga de documento) y **Command** (especifica un comando cargar/evaluar o abrir una herramienta de TPsi). Su especificación detallada se encuentra en el Anexo B.10.

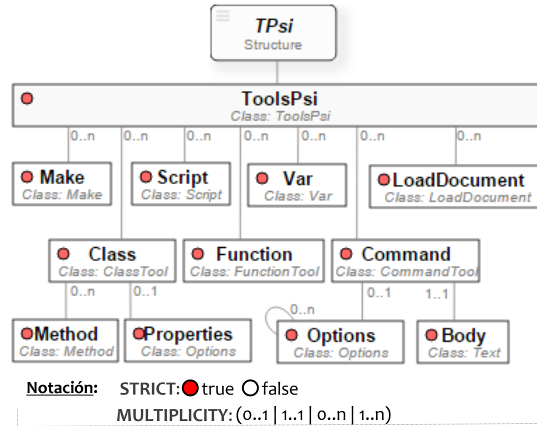


Figura 5-27. Diagrama de Estructura de Utilidades de Programación de TPsi.

5.6.3.2 Diagrama de clases

En la Figura 5-28 se muestra una visión general de los principales subcomponentes, clases y widgets que conforman el diagrama de clases del **Componente TPsi**. El **Kernel** de las herramientas Psi está conformado por: la clase raíz *ToolsPsi* (encargada de administrar todas las herramientas de TPsi), la clase abstracta *Tool* (clase base para todas las TPsi) y el tipo enumerativo *ToolType* (determina el tipo de herramienta), el subcomponente **Generator** (implementa la funcionalidad del Generador de Diagrama) y el subcomponente **Recommendator** (implementa la funcionalidad del Recomendador de Diagramas).

La clase abstracta *ToolsPsiParser* es el analizador de una gramática Psi para un lenguaje Psi. El subcomponente **Help** da soporte a la ayuda y el idioma de las diversas Herramientas Visuales, y, La clase *ContainerManager* es el administrador de contenidos para las Herramientas Visuales.

Por último, los widgets ayudan a implementar las Herramientas Visuales basadas en la definición de Widget de jQuery (jqueryui.com/widget).

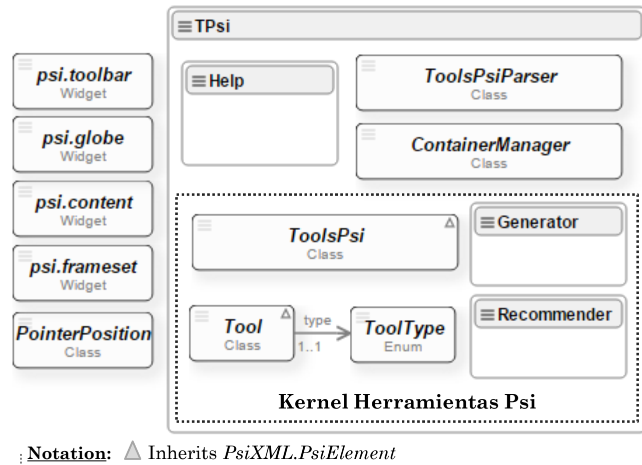
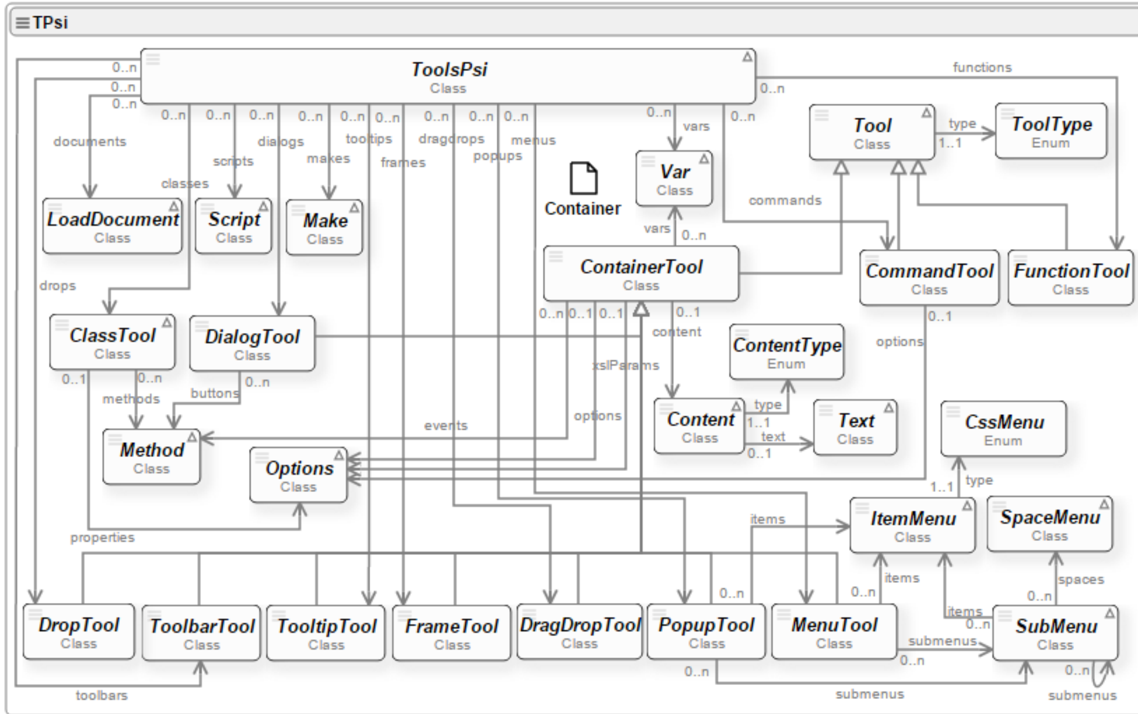


Figura 5-28. Diagrama de clases de los principales elementos de programación del Componente TPsi.

En la Figura 5-29 se muestra el diagrama de clases simplificado para los Utilidades de Programación y las Herramientas Visuales de DPsi.

La clase abstracta *ConteinerTool* desarrolla las definiciones básicas de *Contenido*, *Posición*, *Eventos* y *Opciones* de las Herramientas Visuales. El *Contenido* se apoya en la clase *Content*, la cual define el tipo de contenido a construir, el contenedor de salida y los recursos (fragmentos HTML, ficheros XML, XSL, CSS, etc.). Por ejemplo, la combinación de XML y transformaciones XSLT, requiere que los ficheros XML y XSL hayan sido cargados previamente con *LoadDocument*, o haber especificado las URLs respectivas, o hacer combinación de las dos.



Notation: △ Inherits *PsiXML.PsiElement*

Figura 5-29. Diagrama de clases del Componente TPsi para las Herramientas Visuales y Utilidades de Programación.

5.7 EL GENERADOR DE DIAGRAMAS

Una herramienta importante del lenguaje TPsi es la capacidad de generar diagramas programables en forma automática utilizando reglas y algoritmos basados en las en las DSs heterogéneas XML/JSON. Esta generación se basa en el uso de GLPsi y en la creación de programas en DPsi.

5.7.1 ANÁLISIS Y DISEÑO DEL LENGUAJE GENERATOR PARA TPSI

En la Figura 5-30 se muestra el diagrama de proceso para la generación de un diagrama programable. En el proceso se definen cuatro tareas:

T1. Declarar **Variables** (etiqueta **Var**) y/o **Matrices** (etiqueta **Matrix**).

T2. Generar **Figuras**, **Listas** o **Paneles** (etiqueta **Graphs**).

T3. Generar **Líneas** (etiqueta **Lines**).

T4. Ejecutar un **Proceso** (etiqueta **Process**) para la manipular la posición y el estilo de elementos gráficos generados.

Para Iniciar el proceso se debe tener a disposición un conjunto de elementos gráficos (programas GLPsi) registrados, un administrador del diagrama (programa inicial DPsi que incluye un lienzo SVG).

La declaración de **Variables** o **Matrices** tienen como objetivo facilitar la reubicación de elementos gráficos. Una **Matriz** $\mathrm{M}_{m\times n}$ establece una cuadrícula imaginaria en el canvas del diagrama, con ciertas dimensiones de ancho y alto para cada ítem $\mathrm{m}_{ij} \in \mathrm{M}$. Cada ítem m_{ij} puede admitir un único elemento gráfico que se centra automáticamente. La Matriz se usa para la generación de los diferentes diagramas:

de estados, de bloques, de clases, de procesos de negocios, de colaboración, o de árbol, por ejemplo, como se presentará en el Capítulo 8.

Para generar un conjunto de **Figuras**, **Listas** o **Paneles**, se requiere crear una consulta (con selectores o “queries”) sobre los DSs, elegir un elemento gráfico (ya sea Figura, Lista o Panel) de la GLPsi, clonarlo y ubicarlo en una posición fija o aleatoria, con el administrador de diagramas.

Para generar **Líneas** (con o sin conexión entre elementos gráficos), también se requiere crear una consulta sobre los DSs, elegir una Línea de GLPsi, y ubicarla con el administrador de diagramas. Si la Línea es un conector entre dos elementos gráficos (Figura, Lista o Panel), estos deben existir previamente en el diagrama.

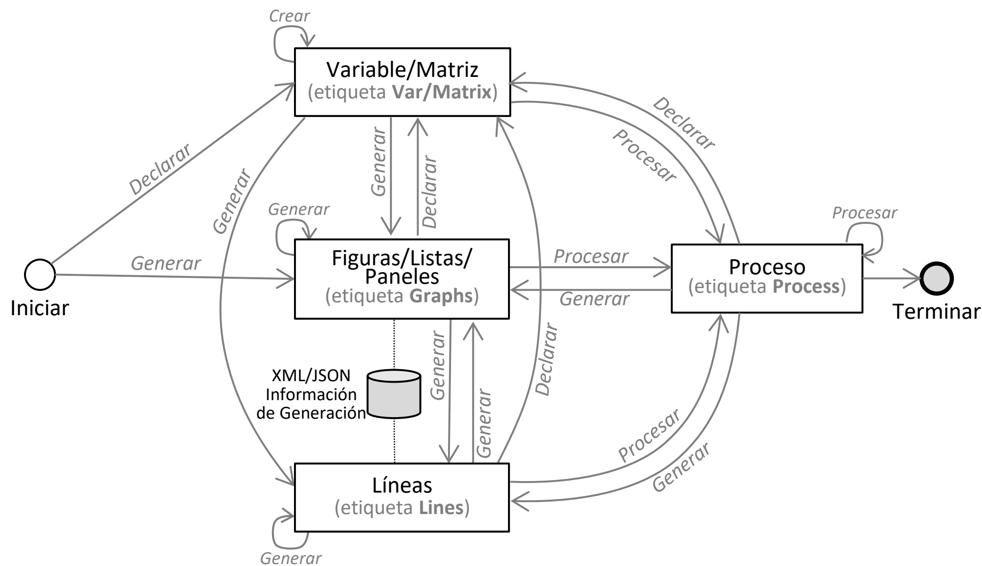


Figura 5-30. Definición conceptual (diagrama de proceso) del Generador de Diagramas.

Un **Proceso** tiene a su disposición los elementos gráficos generados, y con ellos se crean algoritmos para la reubicación de dichos elementos gráficos o para la modificación de su visualización y presentación. El Generador cuenta con un conjunto de algoritmos de reubicación basados en matrices y con información cuantitativa (por ejemplo, número de figuras, de listas, de paneles y de líneas, número de conectores de elementos gráficos, etc.).

El diagrama programable tiene la capacidad de ejecutar diferentes Generadores de Diagramas, brindando una herramienta poderosa para la construcción de DSVLs.

Por último, en el Anexo B.11 se muestra la plantilla para generación de diagramas, etiqueta **Generator** incluido en el lenguaje TPsi.

5.7.2 Diseño del Generator para TPsi

Continuando con la metodología para el diseño de lenguajes Psi, en esta sección se muestra el PsiLSD y el diagrama de clases del Generador de Diagramas de TPsi.

5.7.2.1 El PsiLSD de Generator para TPsi

En la Figura 5-31 se muestra el PsiLSD del Generador de Diagramas. Teniendo en cuenta el proceso de generación de la Figura 5-30, la etiqueta **Generator** establece el inicio de la generación y adiciona el administrador de diagrama. Además, está etiqueta contiene las etiquetas: **Options**, **Var**, **Matrix**, **Graphs**, **Lines** y **Process**. La etiqueta **Options** establece el tipo de algoritmo, estilos a aplicar y configuración para

la generación del diagrama programable. La etiqueta **Var** permite definir una variable, mientras que la etiqueta **Matrix** crea una Matriz $\mathrm{M}_{m \times n}$ donde se define el número columnas *m* y número de filas *n*, y el ancho y alto para cada celda, entre otras opciones.

La generación de elementos gráficos (figuras, nodos, listas o paneles) se realiza mediante la etiqueta **Graphs**. Esta etiqueta establece la búsqueda sobre el DS (XML/JSON), el tipo de posición (aleatorio o fijo) y la capa gráfica (etiqueta **Layer** de DPsi) donde se adicionan los elementos generados. Además cuenta con la etiqueta **Source** (establece una plantilla de texto para el elemento gráfico DPsi) y múltiples etiquetas **Event** (eventos para personalizar los elementos gráficos una vez generados).

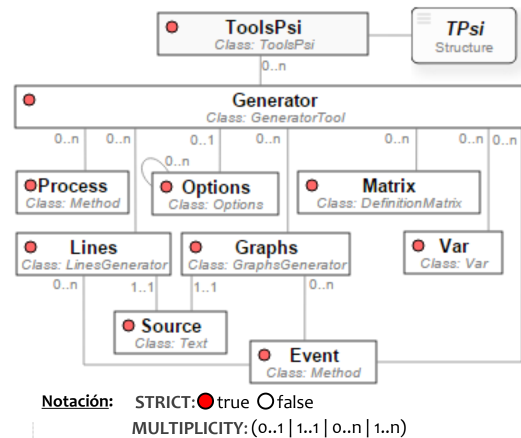


Figura 5-31. Diagrama de Estructura del Generador de Diagramas de TPsi.

Para la generación de líneas o conectores se usa la etiqueta **Lines**. En ella se define la búsqueda del DS (XML/JSON) y la capa gráfica en la que se adicionan las líneas o conectores. Si se generan conectores, se solicita la referencia a los elementos gráficos de inicio y fin. De igual forma que **Graphs**, contiene una etiqueta **Source** y múltiples etiquetas **Event** con la misma funcionalidad. Por último, la etiqueta **Process** contiene la funcionalidad en JavaScript para procesar los elementos gráficos generados por las etiquetas **Graphs** y **Line**. Con esta etiqueta, se pueden establecer modificaciones de elementos, crear nuevos elementos gráficos, usar matrices para su reubicación, crear nuevos algoritmos de diagramación automática, etc. Es decir plantear la estrategia para pintar el diagrama programable.

Nótese que la gramática TPsi para el Generador de Diagramas permite una combinación entre todas las etiquetas mencionadas y nos garantiza el proceso de generación de la Figura 5-30.

5.7.2.2 Diagrama de clases

En la Figura 5-32 se muestra el subcomponente **Generator**. La clase *GeneratorTool* implementa el proceso de generación de diagramas de la Figura 5-30. En esta clase se tienen los atributos para el manejo de variables, opciones, matrices, líneas, elementos gráficos, procesos y los eventos propios de TPsi.

La clase abstracta *BaseGenerator* desarrolla las funcionalidades comunes de las etiquetas **Graphs** y **Lines**, es decir, crea la plantilla para el elemento gráfico DPsi y el manejo de eventos. La clase *LinesGenerator* implementa la generación de líneas basado en la clase base *LineBase* del componente DPsi. Mientras que la clase *GraphsGenerator* implementa la generación de figuras, nodos, listas o contenedores basados en la clase base *GraphBase* del componente DPsi.

La clase *DefinitionMatrix* implementa una matriz para la ubicación de elementos gráficos. Esta clase básicamente está compuesta por una matriz de celdas (atributo *cells*, instancias de *Cell*). La clase *Cell* tiene como atributos el origen, el ancho y el alto, además, tiene la capacidad de determinar las dimensiones de elemento gráfico y centrarlo en la celda.

Por último, la clase *Paths* implementa la búsqueda de caminos entre elementos gráficos a través de sus conectores.

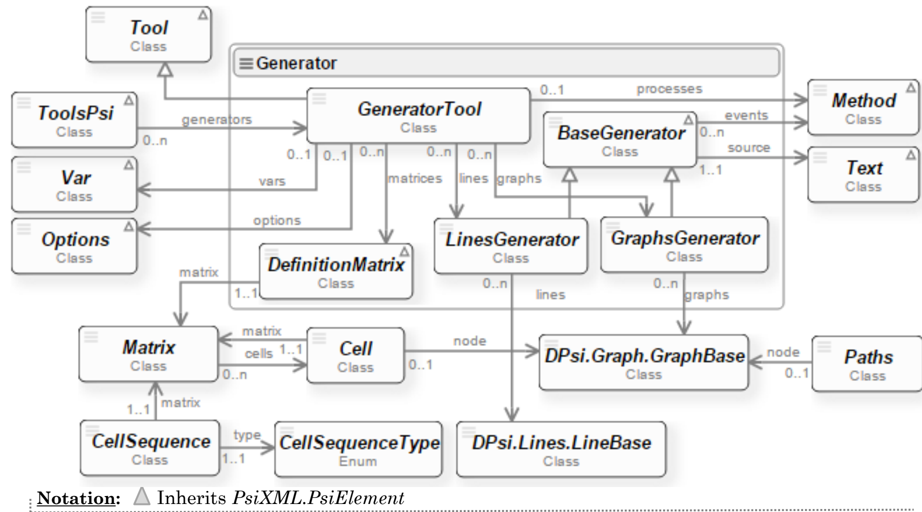


Figura 5-32. Diagrama de clases del Subcomponente Generator del Componente TPsi.

5.8 EL RECOMENDADOR DE DIAGRAMAS

La recomendación de un diagrama se requiere cierta versatilidad y facilidad en la generación de diagramas, como la que se propuso en la sección anterior. En esta sección se establece el lenguaje Psi, que se incluye como herramienta TPsi, para los mecanismos de recomendación de diagramas (O4).

5.8.1 ANÁLISIS Y DISEÑO DEL LENGUAJE GENERATOR PARA TPSI

En la Figura 5-33 se establece el diagrama de proceso para la recomendación de un diagrama programable. En el proceso se definen cuatro tareas:

T1. Definir **Parámetros** (etiqueta **Options**).
T2. Definir **Algoritmo de Recomendación** y utilizando los parámetros (etiqueta **Algorithm**).
T3. **Generación de Diagramas** (etiqueta **Generation**).
T4. **Guardar Recomendación** los resultados para las próximas recomendaciones (etiqueta **Save**).

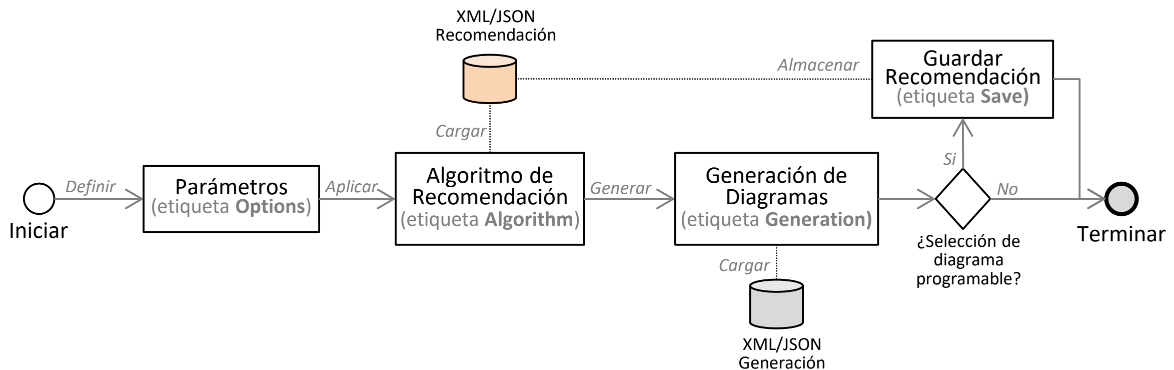


Figura 5-33. Definición conceptual (diagrama de proceso) del Recomendador de Diagramas.

Para iniciar el proceso se debe tener a disposición un conjunto de elementos gráficos (programas GLPsi) registrados, un administrador del diagrama (programa inicial DPsi que incluye un lienzo SVG) y generadores de diagramas (programa TPsi que incluye la etiqueta **Generator**).

Los **Parámetros** definen el tipo de diagrama, el algoritmo de generación, el número de elementos gráficos, las dimensiones del diagrama, etc.

Seguido, se aplica un **Algoritmo de Recomendación** implementado en la herramienta TPsi, o se personaliza un código propio en JavaScript. Para ello se carga un DS (XML/JSON Recomendación, ver Figura 5-33) que conserva la historia de resultados de las recomendaciones anteriores. Con el DS de información de generación (XML/JSON Generación, ver Figura 5-33) y el resultado de evaluar el algoritmo (orden, cantidad de diagramas y parámetros) se ejecuta la **Generación de Diagramas**. Estos diagramas programables se presentan al usuario para que él realice su selección. A partir del diagrama seleccionado por el usuario se **Guarda la Recomendación** para futuras generaciones de diagramas.

Al repetir el proceso en diferentes situaciones y con diferentes usuarios, se recopila información valiosa para obtener cada vez una mejor recomendación. La definición del recomendador en lenguaje TPsi se explica en el Anexo B.11.

5.8.2 Diseño de Generator para TPsi

Continuando con la metodología para el diseño de lenguajes Psi, en esta sección se muestra el PsiLSD y el diagrama de clases del Recomendador de Diagramas de TPsi.

5.8.2.1 El PsiLSD de Recommender para TPsi

La última parte de PsiLSD del Lenguaje DPsi, es el PsiLSD del Recomendador de Diagramas y se muestra en la Figura 5-34. Teniendo en cuenta el proceso de recomendación de la Figura 5-33, la etiqueta **Recommender** establece el inicio de la recomendación de los diagramas programables, y contiene las etiquetas: **Options**, **Algorithm**, **Generation**, **Event** y **Save**.

La etiqueta **Options** establece los parámetros que intervienen en la recomendación y la generación del diagrama programable. La etiqueta **Algorithm** establece el algoritmo a utilizar, el DS de recomendaciones (asociado de forma nativa por el lenguaje Psi) y contiene las etiquetas **Options** y **Body**. Con la etiqueta **Options** se establece el tipo de algoritmo y configuración para la generación de lo diagramas programables. La etiqueta **Body** (opcional) se usa para personalizar el algoritmo de recomendación.

La etiqueta **Generation** define la referencia a un contenedor DOM y los parámetros (etiqueta **Options**) necesarios para la generación de diagramas programables. La etiqueta **Event** personaliza los eventos de la herramienta TPsi.

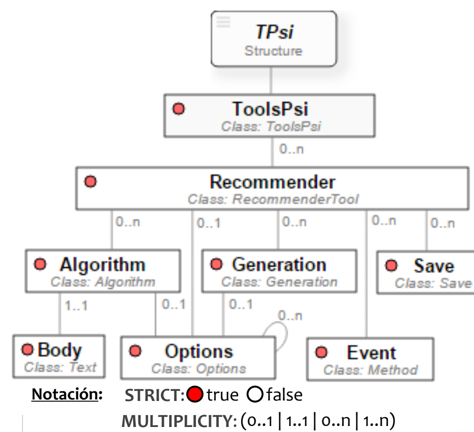


Figura 5-34. Diagrama de Estructura del Recomendador de Diagramas de TPsi.

Por último, la etiqueta **Save** define los parámetros para almacenar la información de recomendación. La generación de elementos gráficos (figuras, nodos, listas o paneles) se realiza mediante la etiqueta **Graphs**. Está etiqueta establece la referencia y búsqueda sobre el DS (XML/JSON), el tipo de posición (aleatorio o fijo) y la capa gráfica (etiqueta **Layer** de DPsi) donde se adicionan los elementos generados. Además cuenta con la etiqueta **Source** (establece una plantilla de texto para el elemento gráfico DPsi) y múltiples etiquetas **Event** (eventos para personalizar los elementos gráficos una vez generados).

5.8.2.2 Diagrama de clases

En la Figura 5-35 se muestra el subcomponente **Recommender**. La clase *RecommenderTool* implementa el proceso de recomendación de diagramas de la Figura 5-33. En esta clase se define los atributos para el manejo de opciones, el algoritmo de recomendación, la generación de los diagramas de recomendación, los eventos propios de TPsi y el almacenamiento para la recomendación.

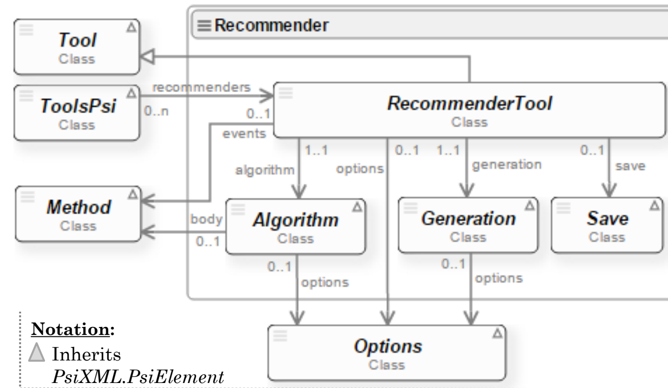


Figura 5-35. Diagrama de clases del Subcomponente Recommender del Componente TPsi.

La clase *Algoritmo* implementa la funcionalidad abstracta de evaluación de un algoritmo de recomendación, con capacidad de personalización mediante una función.

La clase *Generation* ejecuta el Generador de Diagramas con un número de diagramas y las opciones obtenidas después del proceso de recomendación. Además, se tiene a disposición una referencia a un contenedor DOM para generar los diagramas programables en forma de lista, con un botón adicional de cancelación. Si el usuario hace una elección de diagramas, se ejecuta el proceso de almacenamiento definido en el recomendador.

5.9 IMPLEMENTACIÓN DEL COMPONENTE TPSI

En la Tabla 5-3 se muestra la disposición del código fuente y la documentación para el componente TPsi del PsiDiagram. Se incluyen las Herramientas Visuales, las Utilidades de Recomendación, el Generador de Diagramas y el Recomendador de Diagramas.

Tabla 5-3. Implementación y documentación del componente TPsi

Tipo	Descripción
Componente Psi	**DPsi** (Diagramming Psi) http://hilas.ii.uam.es/js/psi/TPsi.js
Documentación online	http://hilas.ii.uam.es/psidiagram (opción “Lenguaje TPsi”)
Diagrama de clases	http://hilas.ii.uam.es/PsiXML/visor.html?title=TPsi&url=source/models/diagram/tpsi/detail.dgl.xml&path=psi/languages/tpsi/&spec=Specification.xml
Proyecto PsiEnvironment	http://hilas.ii.uam.es/PsiXML/index.html?psimodel=TPsi
Github	http://github.com/echavarriaga/PsiDiagram
Otros	Código fuente y documentación en CD Adjunto

5.10 RESUMEN

Un diagrama es un dibujo geométrico y abstracto que representa la información asociada a un problema, fenómeno, proposición o sistema. El concepto de diagrama en esta tesis, incluye como sinónimos

modelos gráficos, lenguajes visuales, bocetos, pinturas, gráficos, etc. Algunos ejemplos diagramas son: diagramas de estados, de bloques, de clases, de Gantt, de Venn, PERT, de procesos de negocios, de entidad-relación, etc.

En este capítulo se ha presentado el Diagrama Programable Psi (PsiDiagram, Programmable Solutions Interpreter Diagram) en un CSWE. Un PsiDiagram es un conjunto de elementos gráficos que define un diagrama basado en lenguaje SVG. Cada elemento gráfico tiene la capacidad de asociar fuentes de información heterogénea (XML/JSON), herramientas visuales y utilidades de programación. En la Figura 5-36 se muestra un diagrama de bloques para la creación de diagramas, diagramas automáticos y/o DSVLs en un CSWE, basados en PsiDiagram.

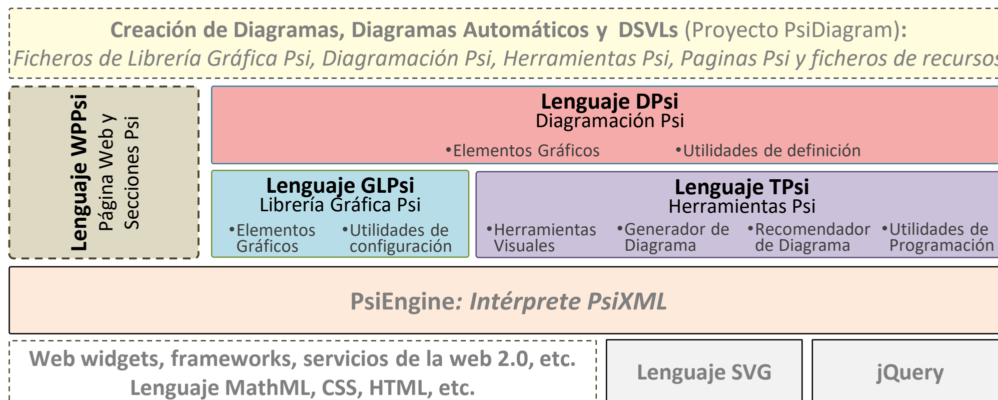


Figura 5-36. Diagrama de bloques para la creación de diagramas, diagramas automáticos y/o DSVLs en un CSWE, basados en PsiDiagram.

El estudio de las diferentes herramientas de diagramación para elaborar diagramas, determinó que se puede construir un PsiDiagram a partir de cuatro elementos gráficos: Figura, Línea, Lista y Contenedor. La Librería Gráfica Psi define estos elementos gráficos usando el Lenguaje GLPsi. La Diagramación Psi es la encargada de tomar elementos de la librería, clonarlos, modificarlos y pintarlos sobre un lienzo SVG mediante el Lenguaje DPsi. Por último, PsiDiagram tiene la capacidad de asociar Herramientas Psi, es decir, asociar herramientas visuales (cajas de diálogo, barras de herramientas, ventanas emergentes, soltar, arrastrar y soltar, menús y menús flotantes), utilidades de programación (inclusión de ficheros y programas Psi, clases, scripts, comandos, funciones y variables), generar diagramas y recomendar estilos de diagramas.

Para cada lenguaje de PsiDiagram (GLPsi, DPsi y TPsi), se hizo un análisis y diseño del lenguaje Psi, y se diseñó e implementó el componente Psi correspondiente, usando la metodología de creación e implementación de lenguajes Psi del Capítulo 3. Además, se afinó el diseño del lenguaje Psi mediante la metodología de diseño para PsiDiagram. La documentación online para todos los componentes se encuentra disponible en la URL http://hilas.ii.uam.es/psidiagram.

Adicionalmente, las Plantillas de Páginas Web Psi (Lenguaje WPPsi) simplifican la creación de una página web a partir de plantillas definiendo secciones de distribución y llamado de contenidos.

Por último, PsiDiagram es usado en el entorno de desarrollo ligero PsiEnvironment, que se explicará en el Capítulo 6, donde se pueden definir Proyectos PsiDiagram. Este entorno proporciona a los programadores simplificar la creación de diagramas, generación de diagramas automáticos y DSVLs.

Capítulo 6. PSIENVIRONMENT: ENTORNO DE DESARROLLO PSI

Este capítulo describe e implementa el entorno de desarrollo **PsiEnvironment** empleando para ello PsiModel (lenguajes MPsi y MIPsi) y PsiDiagram (lenguajes GLPsi, DPsi, TPsi y WPPsi). Este entorno tiene como objetivo la creación de widgets, componentes JavaScript, componentes Psi, diagramas Psi y diagramas programables Psi.

Para abordar este capítulo, se presentará una visión general (sección 6.1), una definición del entorno de desarrollo y los tipos de proyectos (proyectos genéricos, PsiModel y PsiDiagram; sección 6.2). En la sección 6.3, se resumirán las métricas software más reconocidas para el código JavaScript y se definirán las métricas para los proyectos PsiModel y PsiDiagram. En la sección 6.4 se resume como se generan y se muestran estas métricas en el PsiEnvironment.

6.1 INTRODUCCIÓN

El **PsiModel** (Modelo de Programación Psi) descrito en el Capítulo 4, consiste en tener un conjunto de elementos de programación basados en lenguajes Psi en un CSWE para generar código JavaScript y documentación, orientado a la creación de nuevos lenguajes Psi y componentes reutilizables JavaScript. Los elementos de programación separan la especificación (Modelo de Especificación Psi, lenguaje **MPsi**) de su implementación (Modelo de Implementación Psi, lenguaje **MIPsi**). Además, PsiModel dispone de herramientas de transformación y generación de código JavaScript.

Por otro lado, **PsiDiagram** (Diagrama Programable Psi) presentado en el Capítulo 5, describe la creación de un diagrama programable, basados en la Librería Gráfica Psi (lenguaje GLPsi), la Diagramación Psi (lenguaje DPsi), las Herramientas Psi (lenguaje TPsi) y las Páginas Web y Secciones Psi (lenguaje WPPsi).

En este contexto definimos el entorno de desaroollo Psi como:

> **PSIENVIRONMENT**
>
> Es una aplicación web que tiene como objetivo crear un entorno de desarrollo ligero para la edición de Modelos Psi y Diagramas Programables Psi, basados en proyectos y en la edición de ficheros. Hace uso de los lenguajes MPsi, MIPsi, GLPsi, DPsi, TPsi y WPPsi, para su implementación así como del PsiEngine para su ejecución.

Cabe remarcar que PsiEnvironment es una aplicación web y es la herramienta que ha servido para generar todos los resultados y las validaciones de esta tesis.

En la Figura 6-1 se muestra el diagrama de bloques de PsiEnvironment con capacidad de definir proyectos y editar diversos tipos de ficheros (HTML, CSS, JavaScript, Texto, XML, XSLT, etc.), y en particular *editar código escrito en lenguajes PsiModel y PsiDiagram*. Para ello, el PsiEnvironment usa como editor *CodeMirror* (codemirror.net), un framework JavaScript que implementa un editor de texto en el navegador y se especializa en edición de código. Tiene una serie de modos de lenguaje y complementos para facilitar la edición según el tipo de código.

Para nuestro interés, *CodeMirror* tiene la capacidad de código personalizado autocompletable XML mediante un objeto JavaScript configurable. El *PsiXML* al registrar un lenguaje Psi, tiene la capacidad de crear este objeto de configuración para el editor. Es decir, PsiEnvironment dispone de código autocompletado XML para los lenguajes MPsi, MIPsi, GLPsi, DPsi, TPsi y WPPsi (ver Figura 6-2), y en principio para cualquier lenguaje Psi que se implemente.

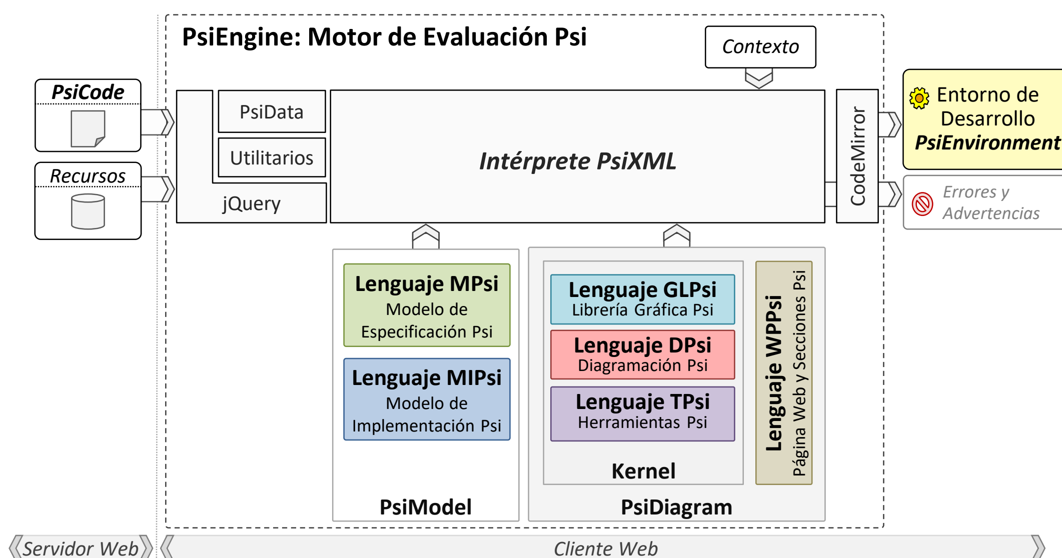


Figura 6-1. Diagrama de bloques de la aplicación web PsiEnvironment.

6.2 ENTORNO DE DESARROLLO

En la Figura 6-2 se muestra una captura de PsiEnvironment con el proyecto del caso de estudio Lenguaje MiniPsi del Capítulo 3. La interface de la aplicación PsiEnvironment está formada por dos secciones: (1) **Menú Principal** (lista de botones en la parte superior de la página que administran proyectos, depuración y redimensionamiento, entre otros) y (2) **Visualización de Proyecto** (conjunto de botones y pestañas en la parte central de la página web que edita un proyecto). En esta sección se resumirán las opciones del menú principal, la definición de un proyecto y los tipos de proyecto (Genérico, PsiModel y PsiDiagram) que soporta PsiEnvironment.

6.2.1 MENÚ PRINCIPAL

En la Figura 6-2 se muestra el Menú Principal de PsiEnvironment, el cual define las siguientes opciones:

- *Inicio*. Muestra la presentación del entorno de desarrollo PsiEnvironment.
- *Proyectos*. Abre la lista de Proyectos Genéricos.

- *Componentes (Psi)*. Abre la lista para Proyectos PsiModel.
- *Diagramas Psi*. Abre la lista de Proyectos PsiDiagram.
- *Depurador*. Abre la ventana de depuración Psi.
- *Redimensionar*. Redimensiona el tamaño de los editores.

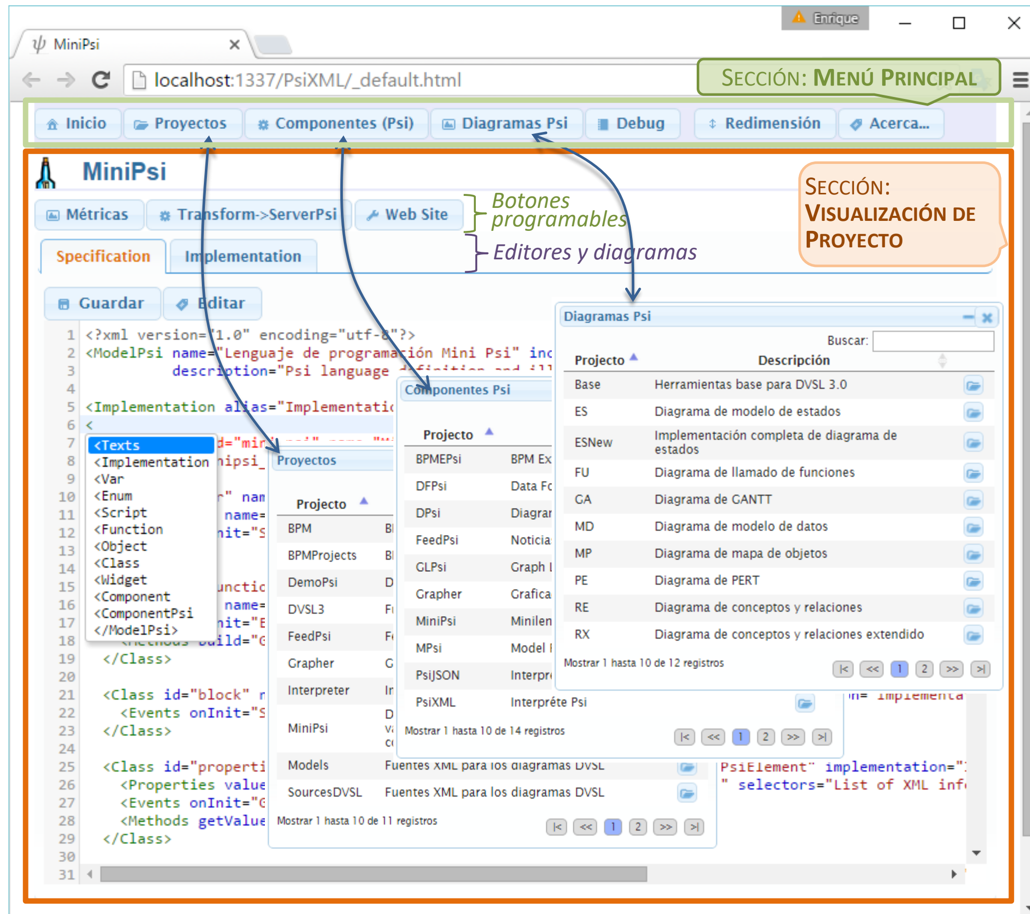


Figura 6-2. Entorno de desarrollo PsiEnvironment.

6.2.2 Definición de Proyecto

El **Proyecto** es un conjunto de ficheros de diversos tipos: MPsi, MIPsi, GLPsi, DPsi, TPsi, WPPsi, Diagramas Psi, HTML, CSS, JavaScript, Texto, XML, SVG, etc. Estos ficheros son almacenados en un directorio del servidor web. La **Visualización de Proyecto** se muestra en la Figura 6-2 y se divide en dos partes:

i. ***Botones programables***: se puede agregar a la visualización del proyecto un conjunto de funcionalidades como: compilar y generar un PsiModel, editar ficheros basados en lenguajes Psi, generar métricas, abrir URLs en una nueva página y guardar ficheros en rutas externas.

ii. ***Editores y diagramas***: es un conjunto de pestañas y en cada una crea un editor *CodeMirror* y edita un fichero del proyecto.

La lista de Proyectos se configura con un fichero XML que sigue la plantilla del Fragmento 6-1. La etiqueta raíz **ProjectsPsi** contiene múltiples etiquetas **Project**. La etiqueta **Project** implementa un proyecto y dispone de un nombre (atributo *name*), una descripción (atributo *description*), un acrónimo (atributo *acronym*), la ruta de la carpeta en el servidor (atributo *path-url*) y la url para guardar ficheros vía AJAX (atributo *save-url*). Cada proyecto contiene *botones programables* y *editores y diagramas*.

FRAGMENTO 6-1. Plantilla XML para definir proyectos genéricos, PsiModel o PsiDiagram.

```
<?xml version="1.0" encoding="utf-8"?>
<ProjectsPsi>
  <!— Lista de proyectos -->
  <Project name="…" description="…" acronym="…" path-url="…" save-url="…">
    <ComponentMetrics name="…" file-key="…" context="…"/> …
    <VisualMetrics name="…" file-key="…" context="…"/> …
    <Transform key="…" name="…" file-key="…" server="…" context="…"/> …   Botones programables
    <Tester key="…" name="…" url="…" data="…"/> …
    <Save key="…" name="…" dir="…" path-url="…" files-keys="…"/> …

    <File key="…" url="…" type="…" tags="…" language="…"/> …               Editores y diagramas
    <Diagram key="…" name="…" url="…" data="…" width="…" height="…"/> …
  </Project>
  …
</ProjectsPsi>
```

Los *botones programables* de un proyecto se definen con las etiquetas:

- **Transform**: crea un botón para la generación de código JavaScript a partir de un modelo de especificación Psi en lenguaje MPsi. Para ello define una clave (atributo *key*), un rótulo para el botón (atributo *name*), la referencia al modelo de especificación Psi (atributo *file-key*) y la ruta y nombre de fichero JavaScript a generar (atributo *server*).
- **Tester**: crea botón con un link a una página web donde se desarrollan las pruebas del proyecto. Dispone de la clave (atributo *key*), el rótulo (atributo *name*), la URL del sitio (atributo *url*) y los parámetros de entrada (atributo *data*).
- **Save**: crea un botón para guardar en otra carpeta una lista de ficheros definidos en el proyecto, usualmente se usa para hacer copias de ficheros del proyecto a la página de prueba. Dispone de la clave (atributo *key*), el rótulo (atributo *name*), el directorio destino (atributo *dir*), un camino relativo del proyecto (atributo *path-url*) y la lista de referencias de ficheros (referencias a etiquetas **File**, atributo *file-keys*).
- **ComponentMetrics**: crea un botón para obtener las métricas para los Proyectos PsiModel. Estas métricas se definen en la sección 6.3.2 y su implementación en PsiEnvironment en la sección 6.4.1.

- **VisualMetrics**: crea un botón para obtener las métricas para los Proyectos PsiDiagram. Estas métricas se definen en la sección 6.3.3 y su implementación en PsiEnvironment en la sección 6.4.2.

Por otra parte, los *editores y diagramas* se definen con las etiquetas:

- **File**: crea en una pestaña un editor *CodeMirror* y abre un fichero del proyecto. Define la clave (atributo *key*), el título de la pestaña (atributo *name*), la ruta del fichero (atributo *url*), el tipo de fichero (atributo *type*), una lista de etiquetas para ficheros XML (opcional, atributo *tags*) y el acrónimo del lenguaje Psi (opcional, atributo *language*). Los tipos de ficheros que soporta actualmente PsiEnvironment son: *html* (página HTML), *css* (estilos CSS), *js* (fichero JavaScript), *text* (fichero de Texto), *xml* (fichero XML), *json* (fichero JSON) y *psi* (programas Psi). Para editar un programa Psi, se debe configurar el atributo *language* con el acrónimo del lenguaje, por ejemplo: la generación de código se usan los lenguajes MPsi (*language*="MPsi") y MPsi (*language*="MIPsi"), o, para el diagrama programable se usa los lenguajes GLPsi (*language*="GLPsi"), DPsi (*language*="DPsi"), TPsi (*language*="TPsi") y WPPsi (*language*="WPPsi").
- **Diagram**: edita en una pestaña el diagrama de clases, el diagrama PsiLSD, o cualquier diagrama implementado con PsiDiagram. Define la clave (atributo *key*), el título de la pestaña (atributo *name*), el editor del diagrama (atributo *url*), el ancho (atributo *width*), el alto (atributo *height*) y los parámetros de entrada (atributo *data*).

Como ejemplo ilustrativo, en el FRAGMENTO 6-2 se muestra el proyecto de tipo PsiModel del lenguaje MiniPsi del Capítulo 3, y es el proyecto visualizado en la Figura 6-2.

FRAGMENTO 6-2. Proyecto MiniPsi para la creación del componente MiniPsi, del caso de estudio MiniPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<ProjectsPsi>
  <!— Proyecto FeedPsi -->
  <Project name="MiniPsi" description="Minilenguaje de programación Psi" acronym="mini"
          path-url="/PsiXML/psi/languages/MiniPsi/" save-url="/psixmlSaveFile">
    <ComponentMetrics key="metrics_minipsi" name="Métricas" title="Métricas MiniPsi"
                      file-key="Specification" context="main.context"/>
    <Transform key="serverpsi" name="Transform->ServerPsi" file-key="Specification"
               server="js\psi\MiniPsi.js" context="main.context"/>
    <Tester key="sample_web" name="Web Site" url="/MiniPsi/index.html" data=""/>
    <File key="Specification" url="Specification.xml" type="psi" language="MPsi"/>
    <File key="Implementation" url="Implementation.xml" type="psi" language="MIPsi"/>
  </Project>
  …
</ProjectsPsi>
```

6.2.3 TIPOS DE PROYECTO

El PsiEnvironment permite crear proyectos de tipo: **Proyecto Genérico**, **Proyecto PsiModel** y **Proyecto PsiDiagram.** Esta sección se dedica a precisar cada uno de estos tipos de proyecto.

La lista de Proyectos Genéricos se define en el fichero "*projects.xml*" del servidor. Se accede a ella con la opción de menú "Proyectos". De igual forma, la lista de Proyectos PsiModel se define en "*components.xml*" accedieéndose en la opción de menú "Componentes (Psi)", y la lista de Proyectos PsiDiagram se define en "*diagrams.xml*" (opción de menú "Diagramas Psi").

6.2.3.1 Proyecto Genérico

En PsiEnvironment un **Proyecto Genérico** ayuda a la administración de todo tipo de recursos y ficheros para la creación de componentes web y/o aplicaciones web. Tiene la capacidad de editar ficheros HTML, CSS, XML, JavaScript, Texto, SVG, entre otros. La URL para la edición de proyectos genéricos es:

http://hilas.ii.uam.es/PsiXML/index.html?**project**=<*name*>, (6-1)

donde <*name*> es el nombre corto del proyecto en la opción de menú "*Proyectos*". En la Figura 6-3 se muestra el diagrama de bloques de un Proyecto Genérico.

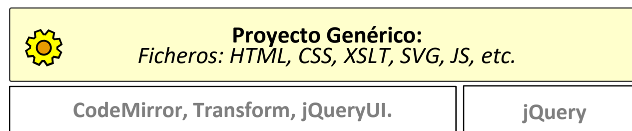


Figura 6-3. Diagrama de bloques de un Proyecto Genérico.

6.2.3.2 Proyecto PsiModel

Un **Proyecto PsiModel** tiene como objetivo crear componentes Psi, componentes JavaScript, aplicación web y/o widgets, basados en el motor de evaluación PsiEngine y en los lenguajes de PsiModel. Este proyecto consiste en crear un conjunto de programas Psi escritos en lenguajes de especificación e implementación Psi (lenguajes MPsi y MIPsi respectivamente) y una lista de *Recursos* (ficheros CSS3, XSL, XML de información y configuración, frameworks JavaScript, widgets, diagramas en SVG, fórmulas en MathML, fragmentos de páginas, solicitudes a servicios web, etc.).

En la Figura 6-4 se muestra diagrama de bloques para un Proyecto PsiModel. Con esta estructura se han desarrollado los 20 casos de estudio PsiModel que se estudiarán y analizarán en la sección 8.2.

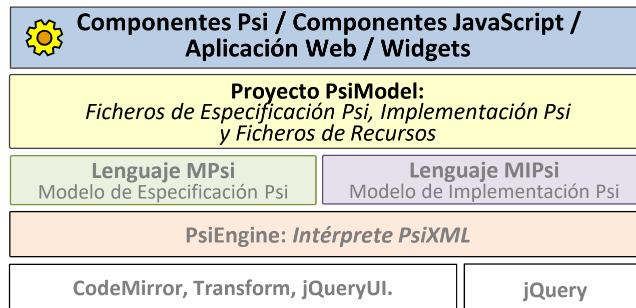


Figura 6-4. Diagrama de bloques de un Proyecto PsiModel.

6.2.3.3 Proyecto PsiDiagram

Un **Proyecto PsiDiagram** administra la creación de diagramas, diagramas de generación automática y DSVLs, basados en los lenguajes de PsiDiagram. Este proyecto consiste en crear un conjunto de programas Psi escritos en lenguajes de GLPsi, DPsi, TPsi y WPPsi y una lista de *Recursos*.

En la Figura 6-5 se muestra el diagrama de bloques de un Proyecto PsiDiagram. Con este tipo de proyecto se han creado todos los diagramas, diagramas automáticos y DSVLs propuestos en esta tesis, principalmente implementados para la Plataforma B2T SCOOP.

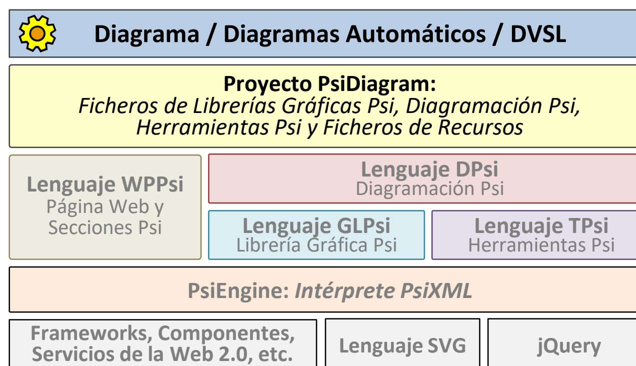


Figura 6-5. Diagrama de bloques de un proyecto de PsiDiagram.

6.3 MÉTRICAS SOFTWARE PARA PROYECTOS DE PSIENVIRONMENT

En el campo de la Ingeniería de Software una *métrica* representa una medida objetiva para conocer o estimar una característica de un sistema de información. Su objetivo es obtener mediciones reducibles y cuantificables para establecer: planificación de presupuestos, estimación de costes, control de calidad, depuración de software, optimización de rendimiento y/o optimización de personal. El software moderno se ha vuelto complejo por la aparición de nuevas tecnologías, de diversos dispositivos, de la globalización y de las redes; y se requieren aplicar técnicas para asegurar la calidad del software como las descritas en (Mistrik, et al., 2015; Galin, 2004; Nicolette, 2015).

Existen una gran cantidad de métricas software en la literatura. En (Riaz, et al., 2009) se hace una revisión sistemática de métricas para la predicción y mantenimiento de software enfocadas a la calidad del software. En (Jatain & Mehta, 2014) se hace otra revisión sistemática sobre métricas y modelos para la fiabilidad del software. En (Tahir & MacDonell, 2012) se presenta un mapeo sistemático sobre las

métricas dinámicas y la calidad de software. Mientras que en (Abílio, et al., 2012) se examina sistemáticamente las métricas para el mantenimiento de software enfocada a la programación orientada a aspectos.

Para un CSWE el desarrollo de aplicaciones web se basa en la creación de páginas web dinámicas, y los lenguajes de scripting como JavaScript determinan su funcionalidad y dinamismo. En este contexto, se requieren métricas software que permitan obtener medidas para valorar las características de la aplicación a nivel cliente web. En (Ahsan, et al., 2012) se rediseñan métricas software para lenguajes basados en prototipos, como lo es lenguaje JavaScript.

En esta sección se resumirá algunas de las métricas software usadas en (Ahsan, et al., 2012) para código JavaScript, y se definirán métricas para los Proyectos PsiModel y Proyectos PsiDiagram. Estas métricas nos permitirán tener herramientas para realizar mediciones y determinar sus principales características. Además, estas métricas software serán utilizadas como parte de la validación de resultados que se presentan en el Capítulo 8.

6.3.1 MÉTRICAS SOFTWARE PARA CÓDIGO JAVASCRIPT

Para un CSWE, el análisis de la complejidad del software se centra en realizar un análisis estadístico de los ficheros JavaScript, o en su defecto, el código JavaScript dentro de una página html. Para nuestro caso, el análisis se realizará sobre el código creado mediante PsiModel en ficheros JavaScript, o la unión de fragmentos en un fichero del código JavaScript insertados en los diferentes lenguajes Psi. Cada fichero JavaScript se denominará **módulo software JavaScript**, o simplemente **módulo software** o **módulo**. En términos generales, un módulo software se define como el código fuente que se vuelca a un fichero físico.

En la Tabla 6-1 se muestra la lista de métricas software para un módulo de software, conjugan una lista de métricas reconocidas como número de líneas de código, número de parámetros, complejidad ciclomática, densidad ciclomática, medidas de complejidad Halstead e índice de mantenibilidad. El resto de esta sección es dedicada a resumir las métricas de la Tabla 6-1.

Tabla 6-1. Lista de métricas software para código JavaScript.

Métrica	**Descripción**	**P**	**T**	**C**
[X]LOC	Número de líneas (S=Código; L=Código Lógico; C=Comentarios; M=Promedio)	X	X	X
NOP	Número de promedio de parámetros	X	X	X
CC[X]	Complejidad ciclomática (N=Número; M=Promedio; D=Densidad)	X	X	X
H[X]	Métricas de complejidad de Halstead (N=Duración del programa; n=Tamaño del vocabulario; V=Volumen del programa; D=Nivel de dificultad; E=Esfuerzo de implementación; T=Tiempo de implementación; B=Estimación de errores)	X	X	X
MI	Índice de mantenibilidad	X	X	X

Nota: **P**: A nivel de Proyecto; **T**: A nivel de Componente; **C**: A nivel de clase.

6.3.1.1 Métricas de número de líneas [X]LOC

El *número de líneas de código* es el recuento del número de líneas de su programa del código fuente. Hay dos tipos de medidas: el *número de líneas físicas* (denotado por SLOC) y el *número de líneas lógicas* (número lógico de estados ejecutables, denotado por LLOC) (Nguyen, et al., 2007). Se utiliza por ejemplo, para estimar la productividad de programación, predecir la cantidad de esfuerzo que se requiere para desarrollar un programa, o estimar el mantenimiento de un sistema de información.

Adicionalmente, el MLOC es el promedio de SLOC de todas las funciones existentes en un módulo software (incluye funciones y métodos de objetos, clases, componentes y componentes Psi). Mientras que el **número de líneas de comentarios** CLOC establece que tan documentado es el código fuente (se deben excluir las líneas de comentarios de código). Usualmente, esta medida es calculada en porcentaje

con respecto a SLOC, y es una medida que se utiliza como indicador de la legibilidad del código, cuanto mayor sea su porcentaje mayor es su documentación, implicando mayor comprensión y en consecuencia, facilidad en mantenimiento. Se pueden establecer los siguientes rangos:

- **<10%** (poco documentado, deficiente),
- **10%-25%** (documentación moderada), y,
- **>25%** (muy documentado).

6.3.1.2 Métrica NOP

Otra medida interesante es el *número de parámetros* de una función NOP. Si este número es grande, se traduce en mayor complejidad, menor entendimiento y peor reutilización. Para determinar el NOP de un objeto o clase, se busca el promedio del NOP de sus métodos. Para un proyecto o componente, se calcula el promedio del NOP de funciones, objetos, clases y subcomponentes.

6.3.1.3 Complejidad ciclomática CC[X]

La *complejidad ciclomática* es una métrica software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Está métrica fue propuesta en (McCabe, 1976) y se basa en el diagrama de flujo de un programa. La complejidad ciclomática (denotado por CCN) define el número de caminos independientes dentro de un fragmento de código para determinar el límite superior del número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez. A partir del análisis de varios proyectos, McCabe encontró que un valor 10 es un límite superior práctico para el tamaño de un módulo.

La complejidad ciclomática es aplicada para el análisis de riesgo en el desarrollo de código, en el mantenimiento de software, en la planificación de pruebas o en la reingeniería. Una posible evaluación del riesgo se resume con los siguientes rangos de CNN:

- **1-5** (función/programa simple, sin mucho riesgo),
- **6-10** (función/programa normal, riesgo normal)
- **11-20** (función/programa de complejidad intermedia, riesgo moderado),
- **21-50** (función/programa complejo, riesgo alto), y,
- **>50** (función/programa poco depurable, riesgo muy alto).

Por otro lado, en (Gill & Kemerer, 1991) muestra la relación entre el CCN y la productividad para el mantenimiento de software estableciendo la *densidad de complejidad ciclomática* (denotado por CCD) como la división entre CCN y el SLOC.

En un módulo JavaScript (fichero de código JavaScript) y para el contexto de PsiModel, se establece la notación CCN para establecer la complejidad ciclomática de las funciones definidas en el módulo. La complejidad ciclomática para objetos y clases (denotada también por CCN), se calcula como el promedio de los CCN de cada método. La complejidad ciclomática para componentes y componentes Psi se calcula como el promedio de los CCN de las funciones, objetos, clases y subcomponentes. La complejidad ciclomática agregada de todo el módulo JavaScript se denotará por CCA.

6.3.1.4 Métricas de complejidad de Halstead H[X]

Las *métricas de complejidad de Halstead* fueron propuestas y diseñadas por (Halstead, 1977). Tienen como objetivo principal determinar una medida cuantitativa de la complejidad de un programa basados en los operados (n_1) y operandos (n_2) distintos y el número total de operandos (N_1) y operadores (N_2) de un módulo. Se definen las siguientes medidas:

- La *Longitud* como $N = N_1 + N_2$ (denotado por HN). La longitud es una medida del tamaño de un programa, entre más grande, mayor será la dificultad de entenderlo.

- El *Vocabulario* como $n = n_1 + n_2$ (denotado por Hn). El vocabulario es la medida complejidad de las sentencias de un programa. Según Halstead, es menos complejo si su número es menor.
- El *Volumen* como $V = N * log_2(n)$ (denotado por HV). El volumen es una medida para dar más peso al número de operadores y operandos distintos. El volumen de una función debe estar en el rango $20 \leq V \leq 1000$. Una función de una línea y sin parámetros es aproximadamente 20. Para $V > 1000$, la función es compleja y tiene mucha funcionalidad asociada. El volumen para módulos se recomienda los limites en un rango de $100 \leq V \leq 8000$.
- El *Nivel de Dificultad* se define como $D = \frac{n_1}{2} * \frac{N_2}{n_2}$ (denotado por HD). Determina el nivel de dificultad de un programa.
- El *Esfuerzo* se define por $E = D * V$ (denotado por HE). El esfuerzo de entender un programa es proporcional al volumen y el nivel de dificultad.
- El *Tiempo* se define por $T = \frac{E}{18}$ seg. (denotado por HT). El tiempo para implementar o entender un programa es proporcional a su esfuerzo.
- El *número de errores estimado* se define por $B = \frac{E^{\frac{2}{3}}}{3000}$ (denotado por HB) y se correlaciona con la complejidad del software. Se recomienda para un módulo que su número de errores estimados debe ser inferior a 2. En caso contrario, el número de pruebas del módulo debe aumentar.

En el contexto de PsiModel se establecen las métricas de Halstead a diferentes niveles: a nivel de funciones, de objetos, de clases, de scripts, de componentes, componentes Psi y por último a nivel de proyecto.

6.3.1.5 Índice de mantenibilidad MI

El *índice de mantenibilidad* (denotado por MI) es una medida que determina la facilidad de mantener un módulo de código fuente. Esta medida fue diseñada en la Universidad de Idaho en 1991 en (Oman, et al., 1991) y se calcula con la siguiente fórmula:

$$\mathrm{MI} = 171 - 3.42 * ln(\mathrm{EM}) - 0.23 * ln(\mathrm{CCM}) - 0.23 * ln(\mathrm{MLOC}),$$

donde EM es el esfuerzo medio de todas las funciones en el módulo, CCM es la complejidad ciclomática media del módulo MLOC es la media de SLOC de sus funciones.

A partir del análisis de esta medida Oman y Hagemeister encontraron que un valor 65 es un límite mínimo para tener una capacidad de mantenimiento para un módulo. Además, se pueden considerar los siguientes rangos para MI:

- **>85** (buena capacidad de mantenimiento),
- **65-85** (mantenibilidad moderada), y,
- **<65** (difícil de mantener, sin comentarios, no estructurado.

En un módulo JavaScript y para el contexto PsiModel, el MI se calcula a nivel de módulo, componente, componente Psi, clase, objeto y función.

6.3.2 Definición de métricas para proyectos PsiModel

El *número de elementos de programación* PE[X] es una medida específica para los proyectos de PsiModel. Se tienen las siguientes medidas: PEV (número de variables), PEF (número de funciones), PEE (número de tipos numerados, PEC (número de clases), PEO (número de objetos), PEW (número de web widgets), PES (número de bloques de código JavaScript).

El *número de componentes* NC[X] en proyecto PsiModel tiene las medidas: CNJ (número de componentes JavaScript) y NCP (Número de componentes Psi).

Por último, se establecen el *número de elementos de una clase* PEC[X] definida en PsiModel. Tiene las siguientes medidas: el número de propiedades (PECP), es número de métodos (PECM) y el tamaño de la clase (PECZ).

En la Tabla 6-2 se lista las métricas software, PE[X], PEC[X] y NC[X] para los elementos de programación Psi.

Tabla 6-2. Lista de métricas software para los elementos de programación Psi.

Métrica	Descripción	P	T	C
PE[X]	Número de elementos de programación (V=Variables; F=Funciones; E=Tipos enumerados; C=Clases; O=Objetos; W=Widgets; S=Script)	X	X	
NC[X]	Número de componentes (J=JavaScript; P=Componente Psi)	X	X	
PEC[X]	Número de elementos de una clase (P=Número de propiedades; M=Número de métodos; Z=Tamaño de la clase)			X

Nota: **P**: A nivel de Proyecto; **T**: A nivel de Componente; **C**: A nivel de clase.

Con la lista de métricas software de software JavaScript, se establece el conjunto total de métricas para un proyecto PsiModel y su implementación. Por facilidad, las métricas se agruparan en: **métricas básicas** (CC[X], NOP, EM, [X]LOC y MI), **métricas de Halstead** (H[X]) y **métricas Psi** (PE[X], PEC[X] y NC[X]).

6.3.3 Definición de métricas para proyectos PsiDiagram

Un Proyecto PsiDiagram es la conjugación de código Psi con código JavaScript. Para tomar medidas aquí, se extrae el código JavaScript en un módulo temporal y se calculan la lista de métricas software de software JavaScript. Para el número de líneas de código, se adiciona el *número de líneas* FLOC, indicando la suma de líneas de código Psi y líneas de código JavaScript se establece el conjunto de métricas.

Para un Proyecto PsiDiagram se pueden obtener las siguientes medidas: el número de *elementos gráficos* GL[X], el *número de herramientas visuales* HV[X], el *número de utilidades de programación* PU[X], el *número de generadores de diagramas Psi* HG y *número total de herramientas y utilidades* Psi NHU. En la Tabla 6-3 se resume la lista de métricas para un proyecto PsiDiagram aplicables a nivel de proyecto (columna “P”), librería gráfica Psi (columna “GL”), diagramación Psi (columna “D”) y herramientas Psi (columna “T”).

Tabla 6-3. Lista de métricas para PsiDiagram.

Métrica	Descripción	P	GL	D	T
[X]LOC	Número de líneas de código (S=Código JavaScript; L=Código Lógico; C=Comentarios; M=Promedio; F=Código Psi+Código JavaScript)	X	X	X	X
GL[X]	Número de elementos gráficos (S=Figura Psi; L=Línea Psi; C=Contenedor Psi; A=Lista Psi)	X	X		
HV[X]	Número de herramientas por visuales (D=Caja de diálogo; B=Barra de herramientas; T=Ventana emergente; F=Marco de contenido; R=Soltar; G=Arrastrar/Soltar; M=Menú; P=Menú flotante)	X			X
PU[X]*	Número de utilidades de programación (V=Variable; F=Función; M=Modificador gráfico; E=Skeleton Psi; C=Clases; L=Cargar de documento; K=Comando Psi)	X	X	X	X
HG	Número de generadores de diagramas Psi	X			X
NHU	Número total de herramientas y utilidades Psi	X			X

Nota: **P**: Proyecto; **GL**: Librería Gráfica; **D**: Diagramación; **T**: Herramientas. (*): **E** (solo en D); **C**, **L** y **K** (solo en H).

Por facilidad, en un Proyecto PsiDiagram se agrupan en **métricas básicas** (CC[X], NOP, EM, [X]LOC y MI), **métricas de Halstead** (H[X]) y **métricas Psi** (GL[X], HV[X], PU[X], HG y NHU).

6.4 GENERACIÓN DE MÉTRICAS SOFTWARE EN PSIENVIRONMENT

Esta sección está dedicada a resumir como es el proceso de la generación y presentación de métricas software para Proyectos PsiModel y PsiDiagram en PsiEnvironment.

6.4.1 PROYECTOS PSIMODEL

Retomando la plantilla para definir proyectos del FRAGMENTO 6-1, en el FRAGMENTO 6-3 se establece la etiqueta **ComponentMetrics** para definir las métricas en un proyecto PsiModel. El botón generado por **ComponentMetrics**, es el encargado de ejecutar el procedimiento para el cálculo de las métricas de un proyecto PsiModel.

FRAGMENTO 6-3. Plantilla XML para definir un proyecto PsiModel.

```
<?xml version="1.0" encoding="utf-8"?>
<ProjectsPsi>
  <!— Lista de proyectos PsiModel -->
  <Project name="…" description="…" acronym="…" path-url="…" save-url="…">
    <Transform key="…" name="…" file-key="…" server="…" context="…"/> …
    <ComponentMetrics key="…" name="…" title="…" context="…"
                      file-key="…" server="…" files-key="…" /> …
    <Tester key="…" name="…" url="…" data="…"/> …
    <Save key="…" name="…" dir="…" path-url="…" files-keys="…"/> …

    <File key="…" url="…" type="…" tags="…" language="…"/> …
    <Diagram key="…" name="…" url="…" data="…" width="…" height="…"/> …
  </Project>
  …
</ProjectsPsi>
```

Generación de Métricas (anotación sobre <ComponentMetrics>)
Botones programables (<Transform>, <ComponentMetrics>, <Tester>, <Save>)
Editores y diagramas (<File>, <Diagram>)

Su funcionamiento es muy simple, hace el mismo proceso de generación de código JavaScript del botón programable **Transform**, pero en vez de guardarlo en el servidor, se envía a un componente para que realice los cálculos de las métricas básicas y de Halstead sobre el código generado, y las métricas Psi se calculan sobre los ficheros Psi fuentes listados en los atributos *key-file* y *key-files*.

En la Figura 6-6 se presenta la captura de la generación de las métricas de un proyecto PsiModel. El reporte mostrado se divide en tres partes: *métricas por proyecto*, *métricas por componentes* y *métricas de los elementos de programación Psi*.

Las *métricas por proyecto* se obtienen sobre los ficheros fuentes MPsi, MIPsi y el fichero generado para el proyecto Psi. Se establecen las métricas básicas en la primera tabla del proyecto y en el diagrama de barras "Número de Líneas" del proyecto. En la segunda tabla se muestran las métricas de Halstead. Las métricas Psi se muestran en diagrama de tarta o circular "Elementos de Programación" del proyecto. Por último, en la tabla de "Compendio de métricas por tipo de elemento de programación Psi" se muestran las métricas básicas y Halstead agrupadas por elemento de programación Psi.

Las *métricas por componentes* (componentes JavaScript y componentes Psi) se calculan generando de modo parcial un módulo para cada componente, y para cada módulo generado se calculan las métricas básicas, Halstead y Psi. Si un componente incluye un subcomponente, para el componente se incluye el subcomponente en el módulo generado para sus cálculos, mientras que para el subcomponente se hace el cálculo como si fuera una componente. En la Figura 6-6, en la parte de métricas por componentes se muestra un diagrama de barras de las métricas Psi y una tabla de compendio de métricas básicas y Halstead agrupadas por componente.

Figura 6-6. Métricas en línea para proyectos PsiModel en PsiEnvironment.

Por último, las *métricas por elementos de programación Psi*, muestran las métricas básicas y Halstead para cada elemento de programación.

Para el cálculo de las métricas, el PsiEnvironment hace uso el paquete **Excomplex** (Stilwell, 2015) implementado sobre NodeJS (nodejs.org). Excomplex hace un análisis de la complejidad del software de

los árboles de sintaxis abstracta (AST) de JavaScript. Por otro lado, para la generación de los diagramas se usó el *framework* Highcharts (highcharts.com).

En PsiEnvironment se simplifica el llamado para abrir un proyecto PsiModel:

http://hilas.ii.uam.es/PsiXML/index.html?**psimodel**=<*name*>, (6-2)

donde <*name*> es el nombre corto del proyecto en la opción de menú "*Componentes (Psi)*".

6.4.2 PROYECTOS PSIDIAGRAM

Retomando nuevamente la plantilla del FRAGMENTO 6-1, en el FRAGMENTO 6-4 se establece la etiqueta **VisualMetrics** para definir las métricas de un proyecto PsiDiagram. El botón generado por **VisualMetrics**, es el encargado de ejecutar el procedimiento para el cálculo de las métricas. Las métricas Psi se calculan sobre los ficheros Psi fuentes listados en el atributo *key-files*.

FRAGMENTO 6-4. Plantilla XML para definir un proyecto PsiDiagram.

```
<?xml version="1.0" encoding="utf-8"?>
<ProjectsPsi>
  <!— Lista de proyectos PsiVisual -->
  <Project name="…" description="…" acronym="…" path-url="…" save-url="…">
    <VisualMetrics key="…" name="…" title="…" context="…"
                   file-key="…" server="…" files-key="…" /> …
    <Tester key="…" name="…" url="…" data="…"/> …
    <Save key="…" name="…" dir="…" path-url="…" files-keys="…"/> …

    <File key="…" url="…" type="…" tags="…" language="…"/> …
  </Project>
  …
</ProjectsPsi>
```


En la Figura 6-6 se presenta la captura de la generación de las métricas de un proyecto PsiDiagram. El reporte mostrado se divide en tres partes: *métricas por proyecto*, *métricas Psi* y *métricas por tipo de herramientas Psi*.

Las *métricas por proyecto* se obtienen a partir de cada fichero generado para cada programa Psi del proyecto obteniendo: el número de elementos gráficos, el número de generadores de diagrama, el número de herramientas, las métricas básicas y las métricas de Halstead.

Las *métricas Psi* se establecen mediante el conteo de las herramientas visuales Psi y las herramientas de programación Psi sobre los programas Psi del proyecto. Además se crea un compendio de todas las herramientas y una comparativa entre herramientas visuales Psi y herramientas de programación Psi.

Por último, las *métricas por tipo de herramienta Psi* establecen las métricas básicas y de Halstead agrupadas por tipo de herramienta Psi, y, las *métricas de diagramación Psi* establece el conteo de elementos gráficos (capas, figuras, líneas, paneles y listas).

Al igual que en los Proyectos PsiModel, para los Proyectos PsiDiagram, el cálculo de las métricas se hace uso el paquete **Excomplex** (Stilwell, 2015) y el *framework* Highcharts (highcharts.com). En PsiEnvironment se simplifica el llamado para la edición de un Proyecto PsiDiagram:

http://hilas.ii.uam.es/PsiXML/index.html?**psidiagram**=<*name*>, (6-3)

donde <*name*> es el nombre corto del proyecto en la opción de menú "*Diagrama (Psi)*".

Figura 6-7. Métricas en línea para proyectos PsiDiagram en PsiEnvironment.

6.5 RESUMEN

La aplicación web **PsiEnvironment** es un entorno de desarrollo ligero para la creación de Modelos de Programación Psi (PsiModel) y Diagramas Programables Psi (PsiDiagram), basados en Proyectos y en la edición de ficheros usando el editor de código *CodeMirror*. En este entorno, un Proyecto es un conjunto de ficheros de diversos tipos almacenados en el servidor: MPsi, MIPsi, GLPsi, DPsi, TPsi, WPPsi, Diagramas Psi, HTML, CSS, JavaScript, Texto, XML, SVG, etc. La interface de PsiEnvironment está formada

por dos secciones: un Menú Principal y la Visualización de Proyecto. La Figura 6-8 se muestra el diagrama de bloques de PsiEnvironment, y se basa en el uso de PsiEngine, CodeMirror, jQuery y en los lenguajes MPsi, MIPsi, GLPsi, DPsi, TPsi y WPPsi.

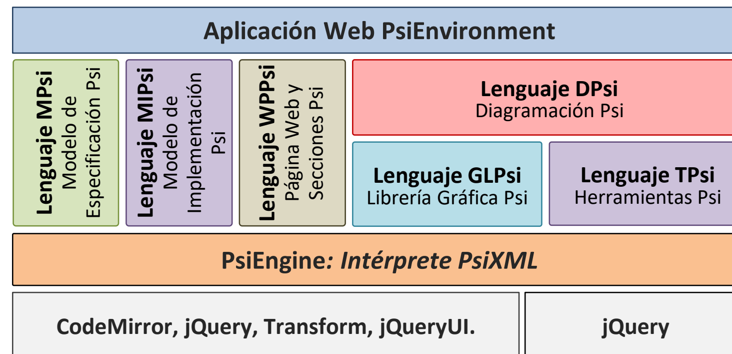


Figura 6-8. Diagrama de bloques del Entorno de Desarrollo PsiEnvironment.

El PsiEnvironment define tres tipos de proyectos: (1) Proyecto Genérico (administra la creación de widgets, componente web y/o aplicaciones web), (2) Proyecto PsiModel (crear componentes y/o componentes Psi, basados en PsiModel) y (3) Proyecto PsiDiagram (crea diagramas, diagramas automáticos gráficos y DSVLs, basados en PsiDiagram).

El PsiEnvironment no sólo permite el desarrollo de proyectos siguiendo las propuestas realizadas en esta tesis, sino también estimar determinadas características del código generado. En la Ingeniería de Software una métrica representa una medida objetiva para conocer o estimar una característica de un sistema de información. Existen una gran cantidad de métricas software en la literatura. En particular, en (Ahsan, et al., 2012) se rediseñan métricas software para lenguajes basados en prototipos, es decir para lenguaje JavaScript. Entre estas métricas están el número de líneas de código, número de parámetros, complejidad ciclomática, densidad ciclomática, medidas de complejidad Halstead e índice de mantenibilidad. Adicionalmente, se definen métricas para Proyectos PsiModel y PsiDiagram. Con todo ello, el PsiEnvironment calcula este conjunto de métricas haciendo uso del paquete Excomplex (Stilwell, 2015) implementado sobre NodeJS (nodejs.org). A su vez, para la presentación de resultados se apoya en el framework Highcharts (highcharts.com).

Cabe resaltar además, que PsiEnvironment ha servido para implementar todos los resultados de esta tesis, y con la ayuda de las métricas software para Proyectos PsiModel y PsiDiagram permiten validar algunas de las características de estos resultados.

Capítulo 7. TRANSFERENCIA TECNOLÓGICA

B2T Concept, S.L.[9] es una compañía española especializada en la gestión de procesos de negocio y gestión del conocimiento para las empresas. En B2T se desarrolló la Plataforma B2T SCOOP (SCOOP) encaminada a la creación de soluciones de negocio basadas en el modelado de Gestión de Procesos de Negocio (BPM, Business Process Management) y en el metamodelado recursivo de negocio, denominado Enterprise Recursive Meta-Modeling (ERMM) (Díez, et al., 2013).

Este capítulo describe cuatro proyectos implementados para la Plataforma B2T SCOOP empleando las propuestas de la presente tesis: Proyecto ADG (generación de diagramas y modelos gráficos automáticos), Proyecto SID (Interfaces de diagrama SCOOP), Proyecto BPME (generación del estándar BPMN 2.0 con PsiDiagram) y Proyecto SMAPPS (creación de aplicaciones móviles basadas en BPME).

7.1 INTRODUCCIÓN: PLATAFORMA B2T SCOOP

La Plataforma B2T SCOOP es una aplicación web que hace uso de la metodología de MDE y contiene dos componentes claves: el *diseñador de modelos heterogéneos* y el *motor de ejecución de modelos*.

El *diseñador de modelos heterogéneos* se utiliza para describir el comportamiento de los procesos de negocio de una empresa creando un Mapa de Procesos a diferentes niveles de modelos como: requisitos de negocio, interfaces de usuario, menús de aplicaciones, acciones del usuario, actores y reglas de seguridad, reglas de negocio, características de integración de sistemas, modelos de datos, diagramas de clases, modelo de estados, entre otros. Con este mapa, SCOOP transforma los modelos heterogéneos en artefactos ejecutables para ser trasladados a un entorno de pruebas y producción.

El *motor de ejecución de modelos* es el encargado de ejecutar los artefactos generados sobre una aplicación web o sobre una aplicación híbrida móvil.

SCOOP posee herramientas para la creación y diseño de modelos heterogéneos basados en DSLs, XML-DSLs, DSVLs, diagramas y modelos gráficos, tales como: diagramas de clases, modelo de estado, modelo entidad relación, estructuras organizativas, administración de proyectos, BPMN v2.0 (OMG, 2011), entre otros. Para dar soporte a la creación de XML-DSLs, DSVLs y modelos gráficos a los diferentes *modelos heterogéneos* creados en SCOOP, se usan PsiModel y PsiDiagram.

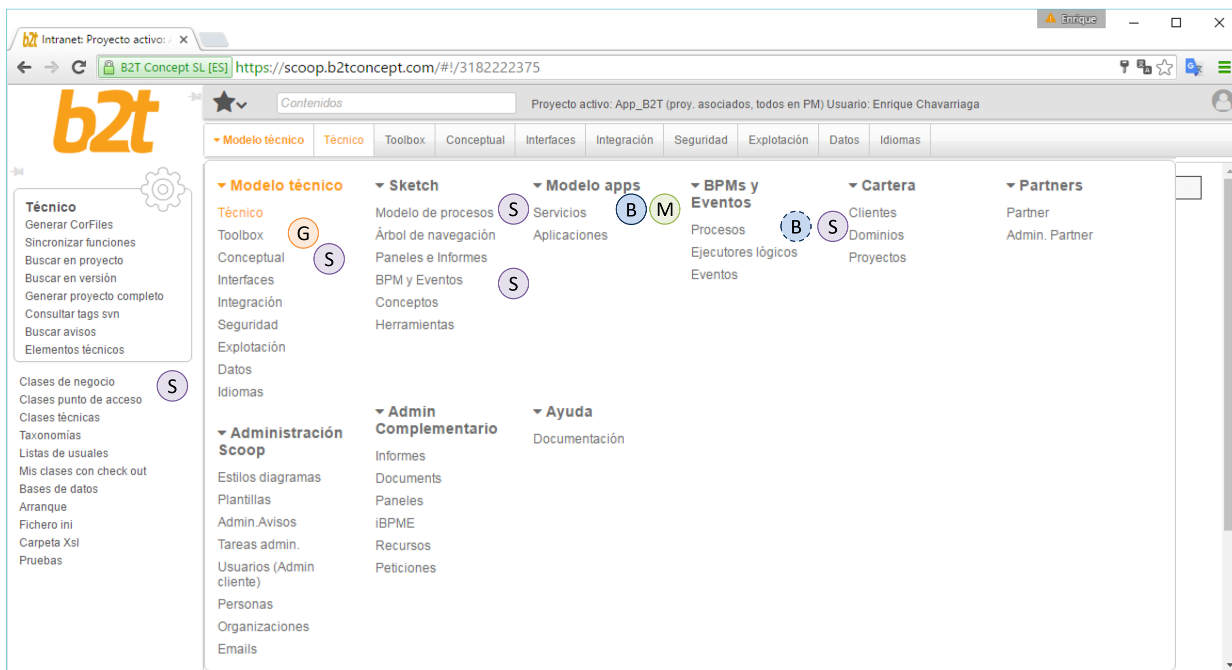
En la Figura 7-1 se presenta una captura de la lista de módulos disponibles en SCOOP. Sobre ella se ubican cuatro proyectos:

---

[9] http://www.b2tconcept.com

- Ⓖ **Proyecto ADG** (Authomatic Diagram Generator): permite la generación de diagramas y modelos gráficos automáticos para SCOOP.
- Ⓢ **Proyecto SID** (SCOOP Interface Diagram): define primitivas en la plataforma SCOOP para la creación y edición de cualquier tipo de diagrama, basado en PsiDiagram.
- Ⓑ **Proyecto BPME** (Business Process Management Engine): implementa la creación y edición de procesos de negocios basado en la especificación BPMN 2.0 (OMG, 2011) y puede ser usado por SCOOP, cualquier aplicación web o widget.
- Ⓜ **Proyecto SMAPPS** (SCOOP Mobile Applications Service): creación de aplicaciones móviles basadas en BPME y servicios SMAPPS.

La sección está dedicada a describir estos proyectos.



Ⓖ **Proyecto ADG**: Generación de diagramas automáticos — Ⓢ **Proyecto SID**: Interfaces de diagrama SCOOP

Ⓑ **Proyecto BPME**: Generación del estándar BPMN 2.0 — Ⓜ **Proyecto SMAPPS**: Creación de aplicaciones móviles basadas en BPME

Figura 7-1. Plataforma B2T SCOOP. Lista de módulos que usan proyectos PsiModel y PsiDiagram.

7.2 PROYECTO ADG

El **Proyecto ADG** (Authomatic Diagram Generator) tiene como objetivo resolver los problemas de visualización de diagramas y de información a partir de fuentes heterogéneas en SCOOP.

El Proyecto ADG es un conjunto de programas Psi y recursos para la generación de diagramas y modelos gráficos de forma automática. Entre los tipos de diagramas implementados están: diagrama de GANT, diagrama de PERT, conceptos y relaciones, diagrama de estados, modelo entidad-relación, llamado de funciones, mapa de objetos y diagrama WBS. Estos diagramas se desarrollaron entre 12/12/2012 a 12/12/2014, para dar el primer paso a un soporte gráfico de algunos componentes de la Plataforma B2T SCOOP.

Cabe destacar que en este proyecto se desarrollaron un conjunto de algoritmos de posicionamiento y de distribución de elementos gráficos basados en caminos y matrices, y que son

utilizados por los otros proyectos relacionados con la generación automática de diagramas.

7.2.1 Estructura del Proyecto

En la Figura 7-2 se muestra el diagrama de bloques del Proyecto ADG, se basa en el diagrama de bloques de proyecto de PsiDiagram de la Figura 6-5, de la sección 6.2.3.3. Tiene dos proyectos base asociados: (1) el **proyecto ADGBase** encargada de crear la funcionalidad y herramientas del diagrama automático, y (2) el **proyecto DSVL3** con interfaz entre SCOOP y el generador de diagramas. Además cuenta con ocho proyectos **TypeADG** para ocho los tipos de diagrama automático.

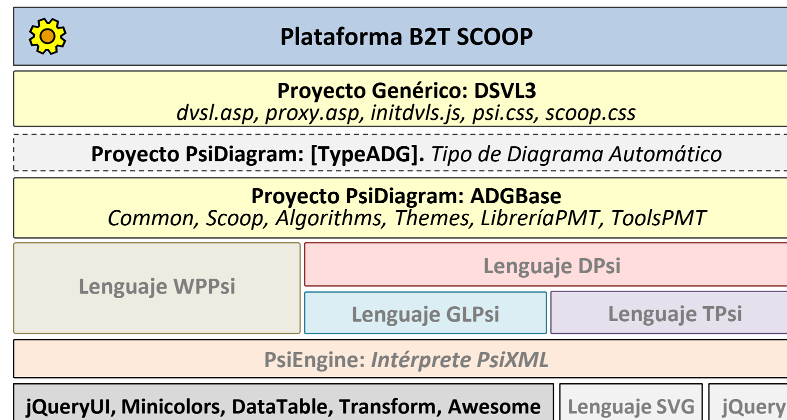


Figura 7-2. Diagrama de bloques del Proyecto ADG.

El funcionamiento del Proyecto ADG es el siguiente: SCOOP solicita un diagrama a *dsvl.asp* la generación de un tipo diagrama automático, *dsvl.asp* carga el *Intérprete PsiXML*, los componentes Psi (GLPsi, DPsi, TPsi y WPPsi), jQuery y los frameworks adicionales para su entorno de trabajo. A continuación, se evalúan los programas Psi del proyecto ADGBase, creando la base visual de los diagramas automáticos, seguido, se evalúan los programas Psi de uno de los un tipo de diagrama automático (proyecto TypeADG).

En la Tabla 7-1 se muestra la lista de proyectos Psi disponibles para la generación de diagramas y modelos gráficos de forma automática. Se divide en dos partes: los proyectos base y los proyectos de tipo de diagrama automático.

Tabla 7-1. Proyectos Psi disponibles para el Proyecto GDA

	Proyecto	Tipo	Nombre	URL(*)
Base	ADGBase	D	Proyecto base ADG	http://hilas.ii.uam.es/project?d=ADGBase
	DSVL3	G	Interface con SCOOP	http://hilas.ii.uam.es/project?g=DSVL3
TypeADG	ADG-ES	D	Modelo de estados	http://hilas.ii.uam.es/project?d=ADG-ES
	ADG-FU	D	Llamado de funciones SCOOP	http://hilas.ii.uam.es/project?d=ADG-FU
	ADG-GA	D	Diagrama de GANTT	http://hilas.ii.uam.es/project?d=ADG-GA
	ADG-MD	D	Modelo de datos	http://hilas.ii.uam.es/project?d=ADG-MD
	ADG-MP	D	Mapa de objetos SCOOP	http://hilas.ii.uam.es/project?d=ADG-MP
	ADG-PE	D	Diagrama de PERT	http://hilas.ii.uam.es/project?d=ADG-PE
	ADG-RE	D	Diagrama conceptos-relaciones	http://hilas.ii.uam.es/project?d=ADG-RE
	ADG-RX	D	Diagrama conceptos-relaciones extendido	http://hilas.ii.uam.es/project?d=ADG-RX
	ADG-WB	D	Diagrama de árbol WBS	http://hilas.ii.uam.es/project?d=ADG-WB

Nota: Tipo: **G** (Proyecto Genérico); **M** (Proyecto PsiModel); **D** (Proyecto PsiDiagram). (*) También disponibles en CD adjunto.

7.2.2 TIPOS DE DIAGRAMAS ADG

De la Figura 7-3 a la Figura 7-9 se muestran ejemplos de los diferentes tipos de diagramas ADG. Cada tipo de diagrama puede soportar más de un estilo, actualmente los siguientes diagramas soportan múltiples estilos: modelos de estados (dinámico, empleo y círculos), modelo de datos (básico, dinámico y campos), diagrama de PERT (completo y bolitas) y diagrama conceptos-relaciones (simple, hogar, tentáculos, estadística).

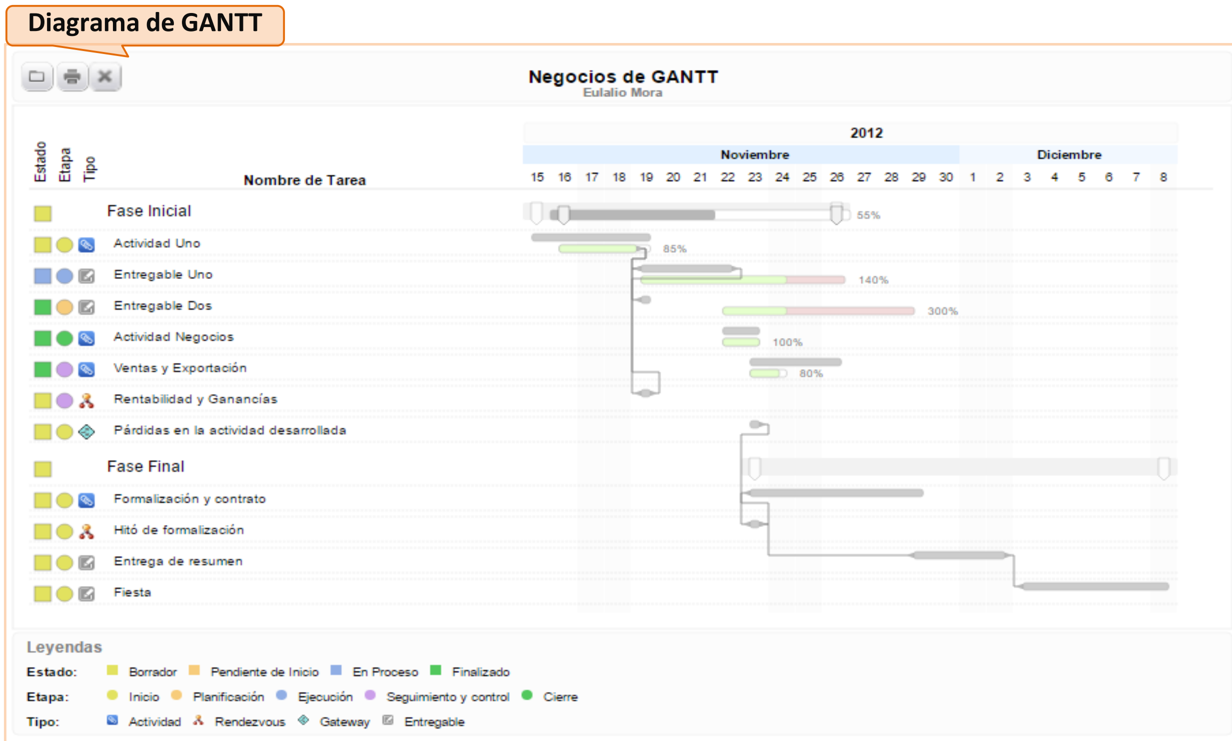


Figura 7-3. Ejemplo de Diagrama de GANTT usando ADG.

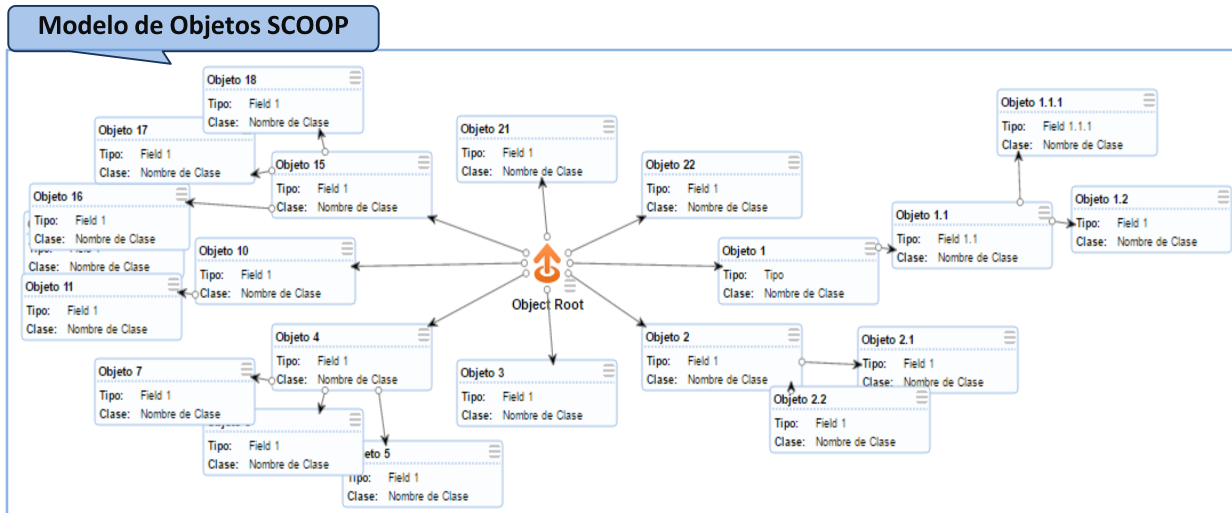


Figura 7-4. Ejemplo de Modelo de Objetos SCOOP usando ADG.

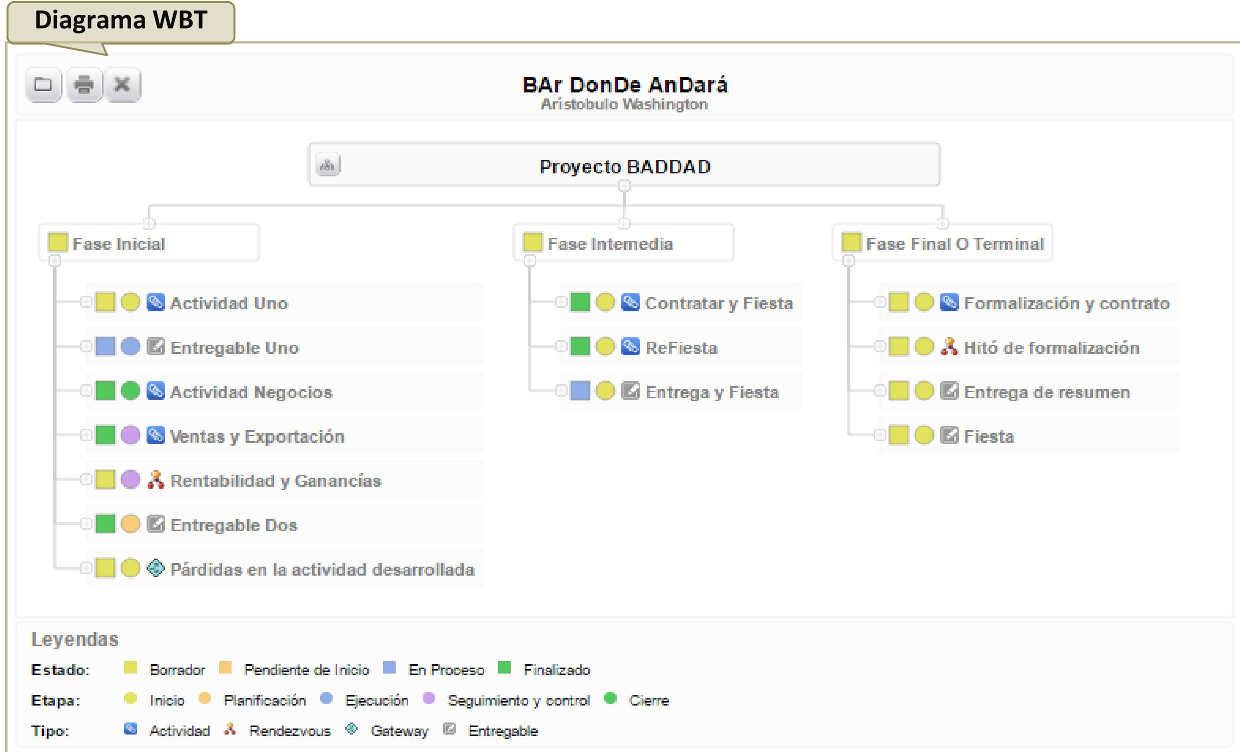


Figura 7-5. Ejemplo de Diagrama WBT usando ADG.

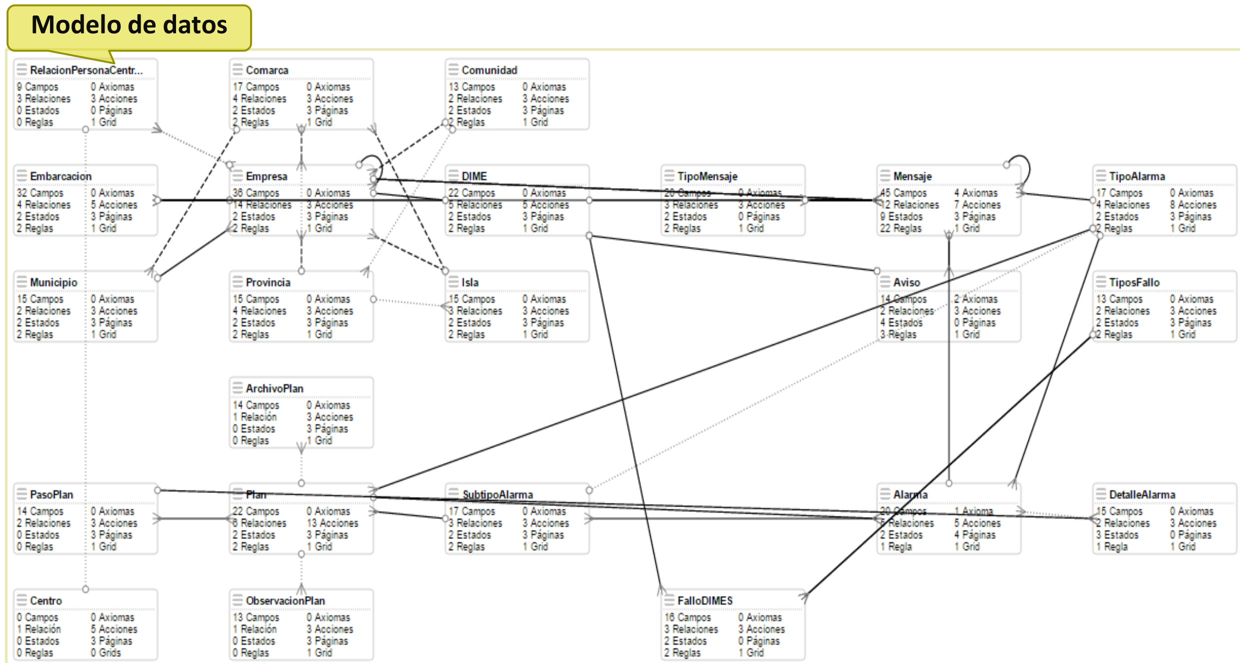


Figura 7-6. Ejemplo de Diagrama Modelo de Datos SCOOP usando ADG.

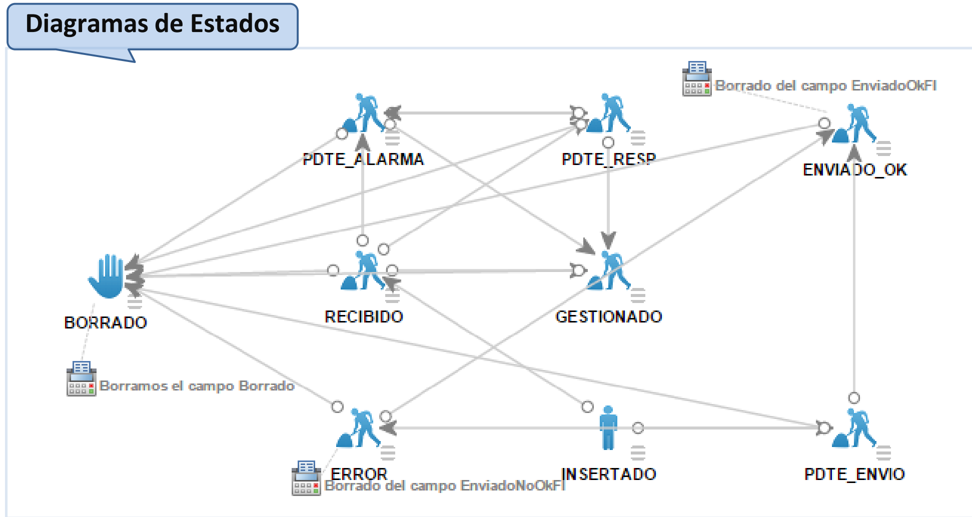


Figura 7-7. Ejemplo de Diagrama de Estados usando ADG.

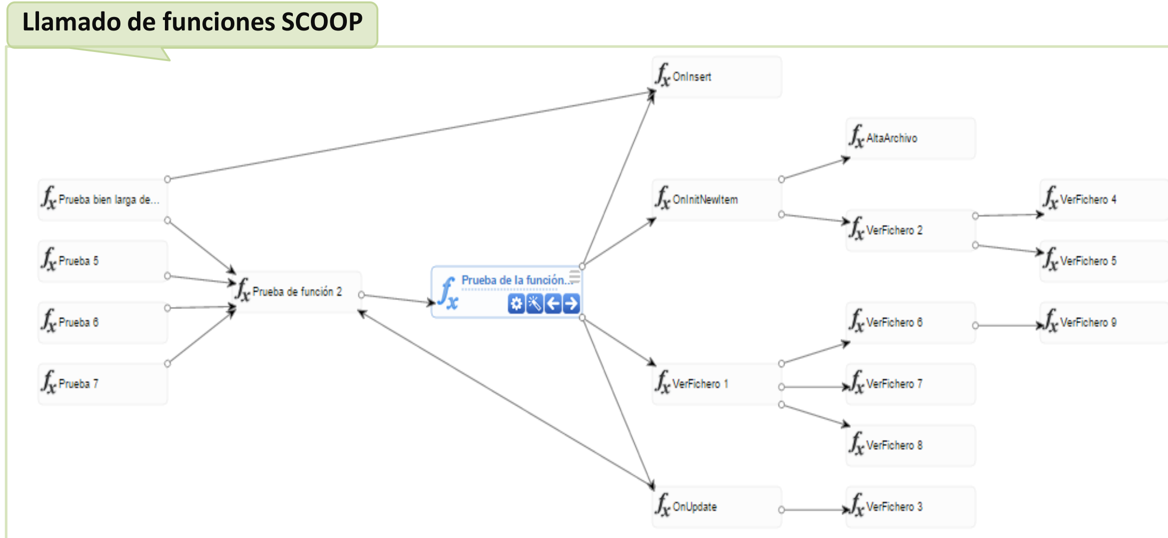


Figura 7-8. Ejemplo de Llamado de Funciones SCOOP usando ADG.

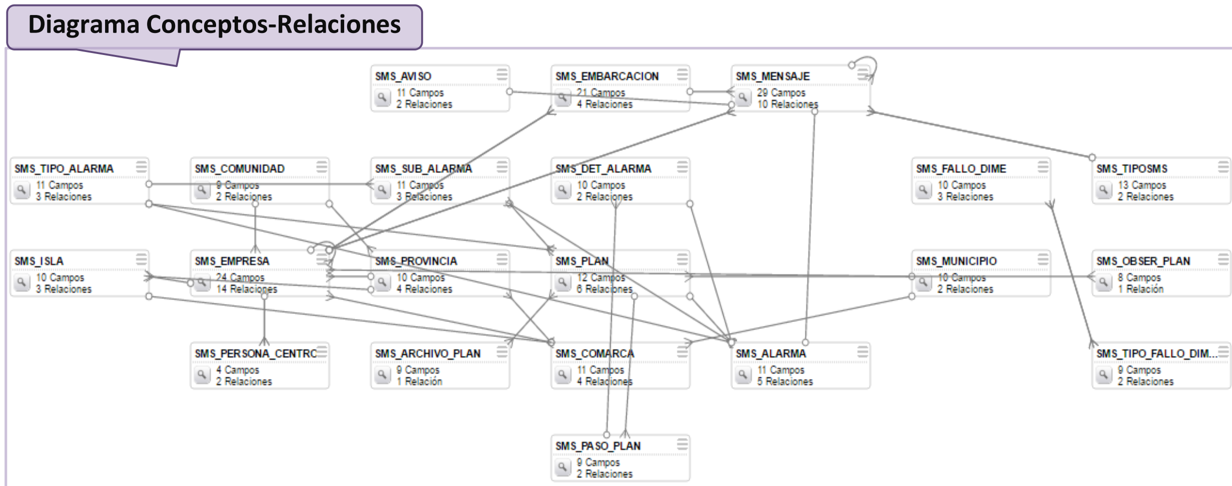


Figura 7-9. Ejemplo de Diagrama de Conceptos-Relaciones usando ADG.

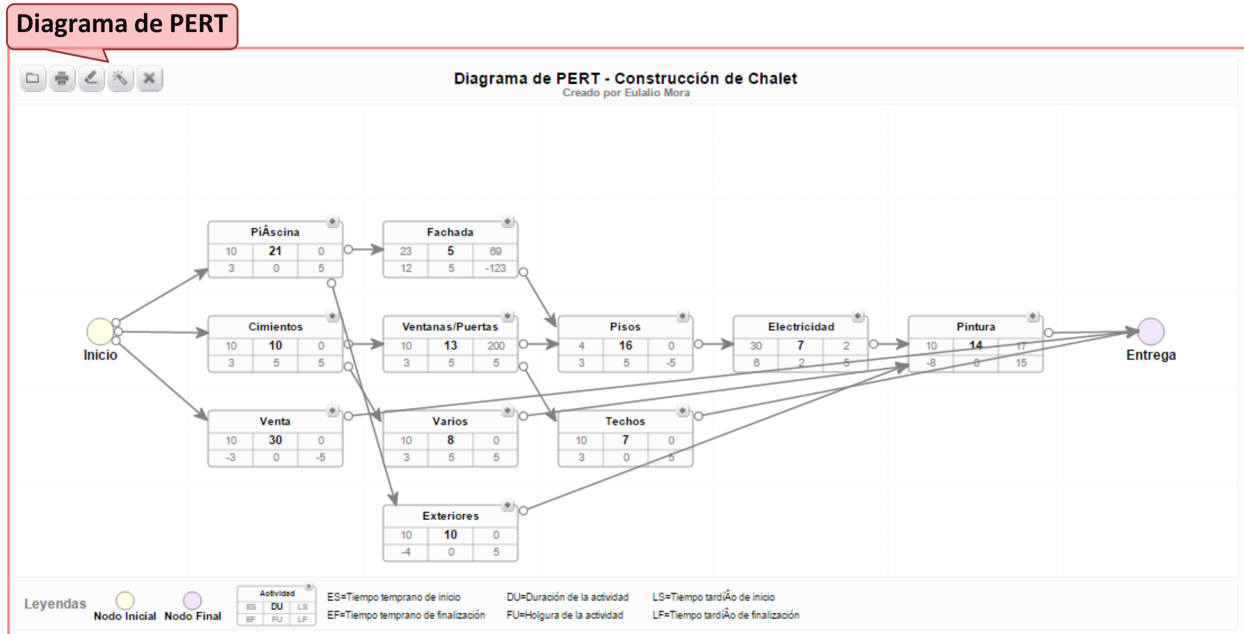


Figura 7-10. Ejemplo de Diagrama PERT usando ADG.

Se creó la aplicación **ADGTest** (proyecto http://hilas.ii.uam.es/project?g=ADGTest) para la visualización de los ejemplos ilustrativos de cada tipo de diagrama. Además, cada proyecto TypeADG dispone de un *botón link* para mostrar un ejemplo de diagrama, si tiene más de un estilo, existe un *botón link* para cada estilo.

7.3 Proyecto SID

El **Proyecto SID** (SCOOP Interface Diagram) es una evolución del Proyecto ADG que cubre la sincronización entre la edición de diagramas y la información preexistente en la plataforma SCOOP. Define primitivas en la Plataforma SCOOP para la creación y edición de cualquier tipo de diagrama, basado en PsiDiagram.

El Proyecto SID es un conjunto de programas Psi para la creación y edición de diagramas en SCOOP, con capacidad de generar automáticamente los diagramas, reutilizando algoritmos del proyecto ADGBase. Entre los tipos de diagramas implementados están:

- Modelo de conceptos-relaciones, y, modelo de estados para el *Modelo Técnico*.
- Modelo de conceptos-relaciones, modelo de estados y modelos de procesos para *Sketch*.
- BPM para *BPMs y Eventos*.

7.3.1 Estructura del Proyecto

En la Figura 7-11 se muestra diagrama de bloques del proyecto SID, se basa en el diagrama de bloques de proyecto de PsiDiagram de la Figura 6-5, de la sección 6.2.3.3. Tiene tres proyectos base asociados: (1) el **proyecto SID** encargado de la creación y edición de diagramas y modelos Psi; (2) el **proyecto DFPsi** es un lenguaje Psi encargado de crear formularios de alta y edición de información asociada al diagrama; y (3) el **proyecto DSVL3** con interfaz entre SCOOP y el generador de diagramas. Además cuenta con seis proyectos **TypeSID** para los cuatro tipos de diagrama SID, en diferentes modelos de SCOOP.

El funcionamiento del Proyecto SID es el siguiente: SCOOP solicita a *dsvl.asp* editar o generar un diagrama, *dsvl.asp* carga el *Intérprete PsiXML*, los componentes Psi (GLPsi, DPsi, TPsi, DFPsi y WPPsi),

jQuery y los frameworks adicionales para su entorno de trabajo. A continuación, se evalúan los programas Psi del Proyecto SID, creando la base visual de creación y edición de nodos y conectores Psi, seguido, se evalúan los programas Psi del proyecto **TypeSID** específicos según corresponda del tipo de diagrama. Además, el Proyecto SID establece la configuración (idioma, tipo de diagrama, herramientas, etc.), las clases y las primitivas que formarán parte de la comunicación entre SCOOP y el diagrama SID. Además, se crean cajas de diálogo genéricas para el alta, la edición, la consulta y el borrado de elementos gráficos, que hace uso de las primitivas según corresponda. El contenido de las cajas de diálogo lo proporciona el lenguaje DFPsi, mediante el uso de plantillas (fichero XML *Template*).

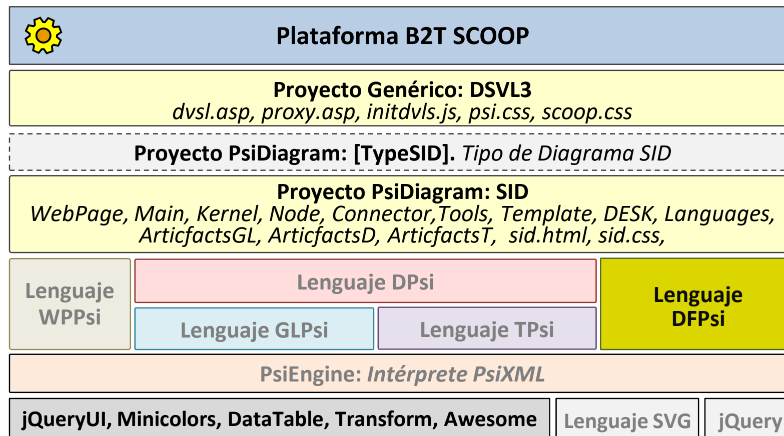


Figura 7-11. Diagrama de bloques del Proyecto SID.

Por último, los programas *ArtifactsGL* (GLPsi), *ArtifactsD* (DPsi) y *ArtifactsT* (TPsi), implementan dos artefactos BPMN 2.0: *anotación de texto* sobre un nodo y *agrupador de elementos*, disponibles en todos los diagramas.

Con esta estructura del proyecto SID, un diagrama de tipo TypeSID solo requiere definir: su librería gráfica GLPsi de nodos y conectores, sus skeletons DPsi, un fichero inicial de diagrama DPsi vacío, un fichero TPsi de configuración de elementos gráfico, y un fichero XML de idiomas.

En la Tabla 7-2 se muestra la lista de proyectos Psi disponibles para la generación de diagramas y modelos gráficos de forma automática. Se divide en dos partes: los proyectos base, los proyectos de tipo de diagrama SID y otros.

Tabla 7-2. Proyectos Psi disponibles para el Proyecto SID

	Proyecto	Tipo	Nombre	URL(*)
Base	SIDBase	D	Proyecto base SID	http://hilas.ii.uam.es/project?d=SIDBase
	DSVL3	G	Interface con SCOOP	http://hilas.ii.uam.es/project?g=DSVL3
	DFPsi	M	Lenguaje DFPsi	http://hilas.ii.uam.es/project?m=DFPsi
TypeSID	SID-CR	D	Modelo de conceptos-relaciones para *Sketch*	http://hilas.ii.uam.es/project?d=SID-CR
	SID-SR	D	Modelo de estados para *Sketch*	http://hilas.ii.uam.es/project?d=SID-SR
	SID-PM	D	Modelo de procesos para *Sketch*	http://hilas.ii.uam.es/project?d=SID-PM
	SID-SCR	D	Modelo de conceptos-relaciones para *Modelo Técnico*	http://hilas.ii.uam.es/project?d=SID-SCR
	SID-SSR	D	Modelo de estados para *Modelo Técnico*	http://hilas.ii.uam.es/project?d=SID-SSR
	SID-SBPM	D	BPM para *BPMs y Eventos*	http://hilas.ii.uam.es/project?d=SID-SBPM
Otros	SIDTest	D	Página de pruebas para crear nuevos diagramas SID	http://hilas.ii.uam.es/project?g=SIDTest

Nota: Tipo: **G** (Proyecto Genérico); **M** (Proyecto PsiModel); **D** (Proyecto PsiDiagram). (*) También disponibles en CD adjunto.

7.3.2 TIPOS DE DIAGRAMA SID Y APLICACIÓN SIDTEST

De la Figura 7-12 a la Figura 7-17 se muestran los diagramas para *Sketch* (modelo de conceptos-relaciones, modelo de estados y modelo de procesos), los diagramas para el *Modelo Técnico* (modelo de conceptos-relaciones y modelo de estados) y el diagrama BPM para *BPMs y Eventos*, todos módulos de SCOOP. Estos diagramas a la fecha de entrega de este trabajo, están en producción.

Para la implementación de un nuevo tipo de diagrama SID, se creó el **proyecto SIDTest**. Es una aplicación de una página en la cual tiene como URL:

http://hilas.ii.uam.es/PsiDiagram/sid.html?**mode**=D&**type**=<*type*>,

donde <*type*> es el TypeSID de diagrama SID. También disponible en el CD adjunto.

Actualmente están disponible para cada tipo de proyecto TypeSID (Tabla 7-2), un ejemplo: el modelo de conceptos-relaciones para *Sketch* (<*type*>=CR), el modelo de estados para *Sketch* (<*type*>=SR), el modelo de procesos para *Sketch* (<*type*>=PM), el modelo de conceptos-relaciones para *Modelo Técnico* (<*type*>=SCR), modelo de estados para *Modelo Técnico* (<*type*>=SSR), y el diagrama de BPM para *BPMs y Eventos* (<*type*>=SBPM).

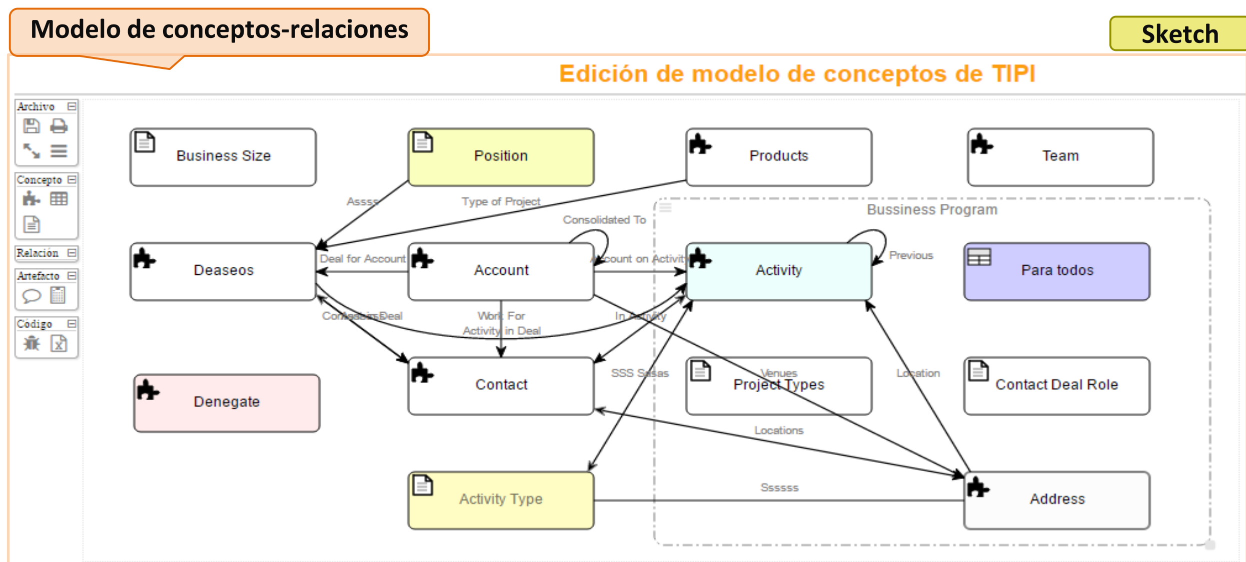


Figura 7-12. Ejemplo de Modelo de Conceptos-Relaciones de *Sketch* usando SID.

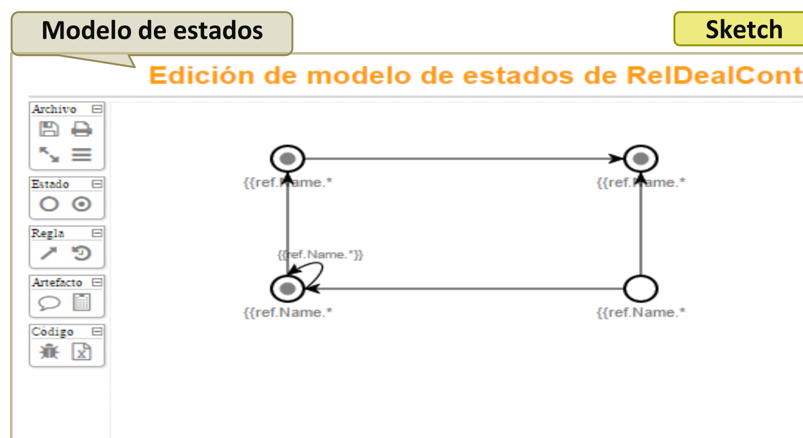


Figura 7-13. Ejemplo de Modelo de Estados de *Sketch* usando SID.

Figura 7-14. Ejemplo de Modelo de Procesos de *Sketch* usando SID.

Figura 7-15. Ejemplo de Modelo de Conceptos-Relaciones de *Modelo Técnico* usando SID.

Figura 7-16. Ejemplo de Modelo de Estados de *Modelo Técnico* usando SID.

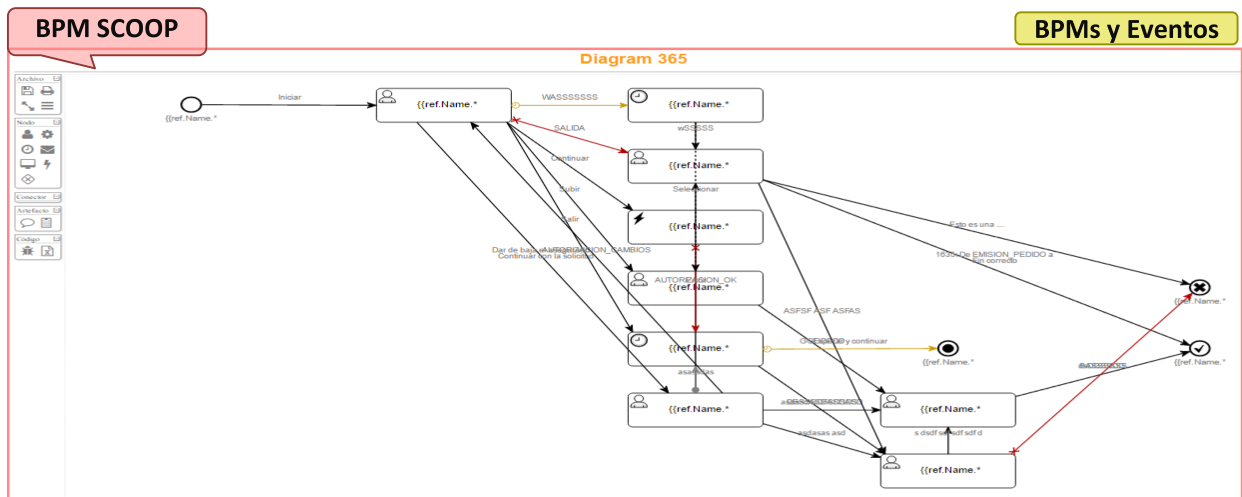


Figura 7-17. Ejemplo de BPM de *BPMs y Eventos* usando SID.

7.4 PROYECTO DE BPME

El **Proyecto BPME** (Bussiness Process Management Engine) tiene como objetivo la creación y edición de procesos de negocios basado en la especificación BPMN 2.0 (OMG, 2011). Este proyecto puede ser usado por SCOOP, aplicación web y/o widget.

El Proyecto BPME es un proyecto que contiene un conjunto de programas Psi que cumple con la especificación de elementos gráficos dispuestos en el estándar BPMN 2.0, y, sirve como base para proyectos que se construyen bajo la filosofía de un proceso de negocios, en otras palabras, es una especie de DSVL genérico para la construcción de procesos de negocios. Los proyectos implementados con esta filosofía son:

- Diagramas SMAPPS, diagramas para la creación de servicios de aplicaciones móviles basados en un subconjunto de BMPN 2.0.
- Generación automática de diagramas BPM para *BPMs y Eventos*.
- Visor de Proyectos BPMEs.

7.4.1 ESTRUCTURA DEL PROYECTO

En la Figura 7-18 se muestra el diagrama de bloques del Proyecto BPME, también se basa en el diagrama de bloques de proyecto de PsiDiagram de la Figura 6-5 de la sección 6.2.3.3. Tiene tres proyectos bases: (1) el **proyecto DFPsi**; (2) el **proyecto BPMEBase** encargado de la creación y edición de diagramas de procesos de negocio basados en BPMN 2.0; y (3) el **proyecto BPMVisor**, proyecto para la visualización de configuración de Tipos de Proyectos BPME TypeBPME (por ejemplo, el Visor de Proyectos BPMEs o los diagramas de Servicios SMAPPS). Además cuenta con dos proyectos **TypeBPME** diferentes modelos de SCOOP.

El funcionamiento del Proyecto BPME es el siguiente: una aplicación (SCOOP, widgets y/o aplicación web) define un tipo de proyecto TypeBPME, el cual debe cargar el *Intérprete PsiXML*, los componentes Psi (GLPsi, DPsi, TPsi, DFPsi y WPPsi), jQuery y los frameworks adicionales para su entorno de trabajo. A continuación, se evalúan los programas Psi del Proyecto BPMEBase, creando la base visual para la administración de elementos gráficos BPMN 2.0, seguido, se evalúan la configuración y las herramientas asociadas (por ejemplo, en el proyecto BPMVisor se encuentran *SampleConfig* y *SampleTools*).

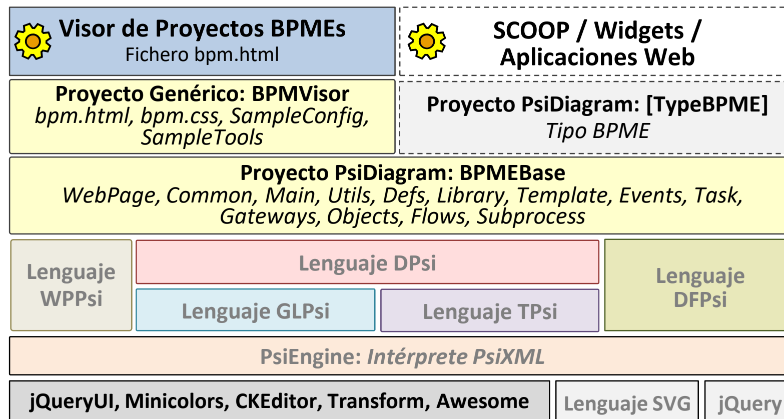


Figura 7-18. Diagrama de bloques de proyectos BPME.

En el proyecto BPMEBase se definen trece programas Psi para la creación y edición de BPMN 2.0. El programa WPPsi *WebPage* es el encargado de la distribución de la página (título, barra de herramientas y diagrama). El programa TPsi *Main* define un conjunto de clases para la administración de un diagrama de proceso de negocio y la comunicación entre la aplicación solicitante (por ejemplo SCOOP o BPMVisor), y también se encarga de ejecutar la configuración particular del proyecto TypeBPME. En particular, define la clase base *BaseElement* que dispone de la funcionalidad genérica para cualquier elemento gráfico BPMN.

El programa TPsi *Common* define herramientas para redimensionar el diagrama, impresión, cajas de diálogos para la creación, edición, eliminación y configuración de los elementos gráficos BPMN, entre otros. Mientras que el programa TPsi *Utils* define herramientas visuales adicionales para el funcionamiento del diagrama. Con el fichero SVG *Defs* se especifican los recursos SVG para los elementos gráficos, mientras que el programa GLPsi *Library* define todos los elementos gráficos BPMN 2.0.

Los programas TPsi *Events*, *Tasks*, *Gateways*, *Objects*, *Flows* y *Subprocess* implementan las funcionalidades particulares de los elementos gráficos *eventos* (iniciales, intermedios y finales), *tareas* (servicio, envió y recibido de mensajes, usuario, grupo, manual, etc.), *gateways* (exclusivo, inclusivo, complejo, paralelo, etc.), *objetos y artefactos* (base de datos, entrada, salida, almacenamiento local, agrupación, mensaje, etc.), *flujos* (normal, condicional, defecto, error, timer, asociación, etc.) y *subprocesos* (simple, loop, instancia múltiple, compensación y ad-doc), respectivamente de los elementos del BPMN 2.0. Todos estos elementos heredan de la clase *BaseElement*.

En la Tabla 7-2 se muestra la lista de proyectos Psi disponibles en BPME y se divide en dos partes: los proyectos base y los proyectos de tipo TypeBPME.

Tabla 7-3. Proyectos Psi disponibles para el Proyecto BPME

	Proyecto	Tipo	Nombre	URL(*)
Base	BPMEBase	D	Proyecto base BPME	http://hilas.ii.uam.es/project?d=BPMEBase
	BPMVisor	G	Visor de BPME	http://hilas.ii.uam.es/project?g=DSVL3
	DFPsi	M	Lenguaje DFPsi	http://hilas.ii.uam.es/project?m=DFPsi
Type BPME	BPMSCOOP	D	Generación de diagramas BPM para *BPMs y Eventos*	http://hilas.ii.uam.es/project?d=BPMSCOOP

Nota: Tipo: **G** (Proyecto Genérico); **M** (Proyecto PsiModel); **D** (Proyecto PsiDiagram). (*) También disponibles en CD adjunto.

7.4.2 Visor de Proyectos BPMEs

El Visor de Proyectos BPMEs es una aplicación web de una página que tiene como objetivo el de implementar proyectos adjuntos (por ejemplo, diagramas de servicios SMAPPS) al Proyecto BPME (ver sección 7.4.1). Para ello se creó el **proyecto BPMVisor** (ver Tabla 7-3). Esta aplicación tiene como URL:

http://hilas.ii.uam.es/PsiDiagram/bpm.html?**bpm**=<*project*>&**process**=<*process*>, (7-1)

donde <*project*> es el nombre de proyecto y <*process*> es el nombre de proceso. Adicionalmente, BPMVisor administra la configuración de la plantilla de nuevo diagrama (programa DPsi *NewDiagram*), nuevo proceso (programa BPMEPsi *NewProcess*, ver sección 7.5.4) y nuevos datos adjuntos (fichero XML *NewData*).

En la Figura 7-23 se muestra un ejemplo ilustrativo en el proyecto BPMVisor (con argumentos bpm=sample&process=test).

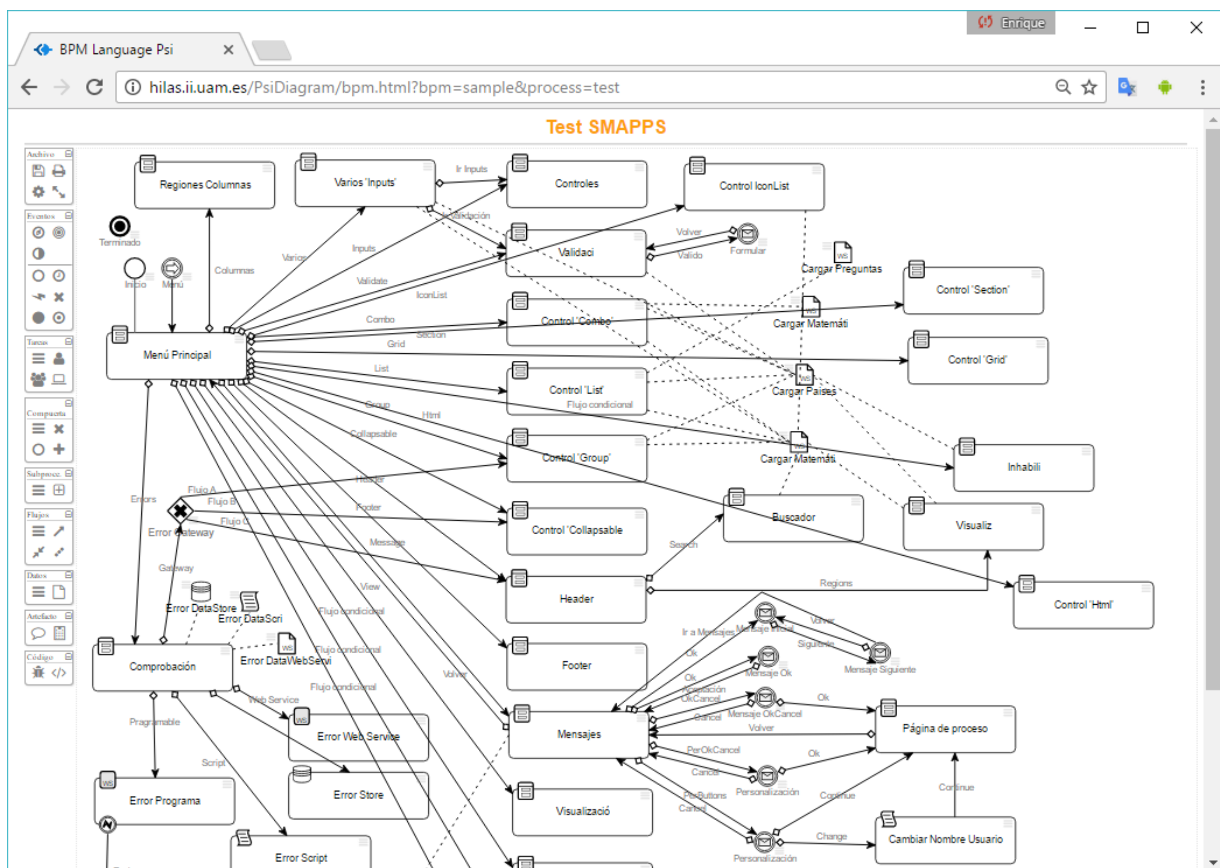


Figura 7-19. Ejemplo del Visor de Proyectos BPME.

Este ejemplo muestra de todos los elementos gráficos de BPMN 2.0 disponibles en BPME. Tiene los ficheros de configuración: el fichero *SampleConfig* (fichero XML) donde están definidos todos los elementos BPMN 2.0, y el fichero *SampleTool* (programa TPsi) donde contiene el menú principal, los menús flotantes y el marco de contenido para personalizar el encabezado del diagrama.

7.4.3 Generación automática de diagramas BPM para BPMs y Eventos

La generación automática de diagramas BPM para *BPMs y Eventos* de SCOOP (proyecto **BPMSCOOP**) fue un proyecto temporal en SCOPP para generar de forma automática el diagrama BPMN basado en un

programa BPMEPsi (ver sección 7.5.4). El funcionamiento era sencillo, SCOOP generaba el programa BPMEPsi y con la herramienta Psi (programas TPsi *SCOOPGenerate* y *SCOOPPaths*) del proyecto BPMSCOOP, se crearon los algoritmos necesarios usando técnicas del recorrido del flujo de procesos de izquierda-derecha, arriba-abajo y curveado de flujos para generar el diagrama. Este algoritmo se basa un nodo inicial con múltiples finales. Esta funcionalidad fue reestructurada para usarse en el proyecto SID-SBPM, dotándolo con la capacidad de generación automática de diagramas BPM.

7.5 Proyecto SMAPPS

El **Proyecto SMAPPS** (SCOOP Mobile Applications Service) tiene como objetivo:

> **Objetivo SMAPPS**:
> Construir un conjunto de componentes y un lenguaje Psi que den soporte a aplicaciones móviles basados en Servicios SMAPPS de B2T SCOOP.

Un **Servicio SMAPPS** se especifica mediante un proceso de negocios usando una versión reducida y especializada de BPMN 2.0, para modelar la navegación entre páginas, la renderización de páginas, la ejecución de script, la conexión con servicios web y el intercambio de información entre Servicios SMAPPS. En otras palabras, el Proyecto SMAPPS consiste en generar el código de una aplicación móvil a partir de la definición de Servicios SMAPPS de SCOOP (DSVLs y fuentes heterogéneas de datos).

Esta sección describirá el funcionamiento de aplicación móvil basada en los Servicios SCOOP, como crear un Servicio SMAPPS, y como definir una aplicación móvil hibrida basado en los Servicios SMAPPS y en el lenguaje BPMEPsi con Material Angular.

7.5.1 Funcionamiento de una aplicación móvil basada en Servicios SCOOP

En la Figura 7-20 se muestra el diagrama para la creación de aplicaciones móviles basadas en Servicios SMAPPS de SCOOP. En la parte izquierda de la figura, el módulo *Modelo Apps* de SCOOP tiene como objetivo especificar, agrupar y administrar los Servicios SMAPPS.

Un **Servicio SMAPPS** se puede ver como un conjunto de ***tareas*** (*páginas web*, *scripts*, *servicios web* y *almacenamientos locales*) y su lógica de navegación, como se muestra en el **diagrama BPME** (ver Figura 7-20). El diagrama BPME se crea mediante el **proyecto BPMESMAPPS** que define la configuración y utilitarios de un proyecto BPME, es decir, crear un subconjunto de elementos gráficos BPMN 2.0 para modelar el Servicio SMAPPS.

Para cada ***tarea*** se especifica su contenido: una tarea *página web*, se especifican los controles (de entrada, selección múltiple, fotos, combos, botones, regiones, etc.) que la componen, la asociación de información, etc.; para una tarea de *servicio web*, se especifican la URL y la estructura de envió y de respuesta, y dónde se va a almacenar; para una tarea de *script*, se especifica el cuerpo del script; y una tarea de *almacenamiento*, se especifica la información que se va a guardar.

Adicionalmente, existe el concepto de *pizarra* que es la estructura de información del servicio SMAPPS, y permite el intercambio de información entre servicios y tareas de la aplicación; y el concepto de *mensajes*, que es una caja de diálogo que se muestran en la ejecución del servicio (puede contener controles de edición). La especificación de todas las *tareas* y de la *pizarra* se hace en un fichero JSON (*data/mapp.json*).

El objetivo principal del diagrama BPME es generar un fichero (*data/process.xml*) que será el código fuente de un programa Psi escrito bajo el lenguaje BPMEPsi. Este lenguaje se explicará en la sección 7.5.4.

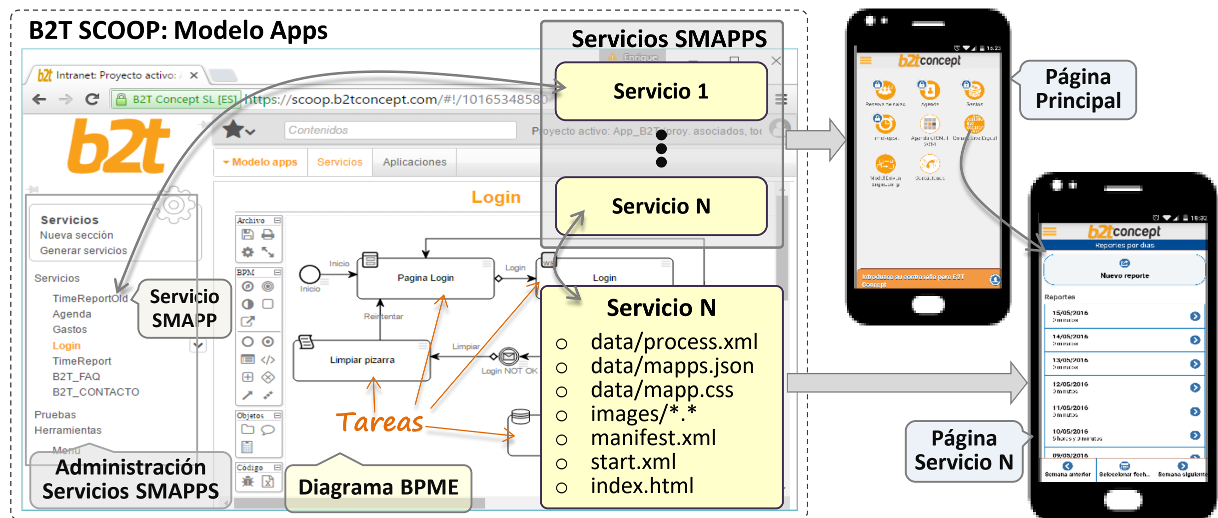


Figura 7-20. Diagrama de creación de una aplicación móvil basada en servicios SMAPPS de B2T SCOOP.

Además, un Servicio SMAPPS lo compone: el *CSS* (*data/mapps.css*); las *figuras* (images/*.*); el *manifiesto* (*manifest.xml*), que determina la configuración con cordova y el dispositivo móvil; y la página de inicio (*index.html*), que crea la infraestructura para la ejecución del servicio SMAPPS.

Una vez se tiene diseñado e implementado los Servicios SMAPPS de una aplicación móvil, el *Modelo Apps* de SCOOP tiene la opción de generar todos los servicios y estos son comprimidos en un fichero *services.zip*. B2T Concept desarrolló una arquitectura para iOS y Android, basada en la filosofía de una aplicación híbrida usando el framework *cordova* (http://cordova.apache.org/), con la capacidad de interpretar y ejecutar los servicios SMAPPS.

En la derecha de la Figura 7-20, la página principal de la aplicación móvil se obtiene recorriendo todos los Servicios SMAPPS, y para cada servicio, leyendo el *manifiesto* para crear su icono de acceso. Adicionalmente se implementó un sistema de validación de usuarios, de tal forma que se habilitan aquellos servicios que estén configurados para dicho usuario. Cuando se entra a un servicio (clic sobre el icono de servicio), se da inicio a la ejecución del programa BPMPsi. En la sección 7.5.4 se resume el Lenguaje BPMEPsi y los componentes AngularSMAPPS que dan soporte a la ejecución del programa BPMPsi. La versión inicial de BPMEPsi fue creada con componentes jQuery Mobile para el soporte visual de la aplicación.

7.5.2 Servicios SMAPPS

Un **Servicio SMAPPS** se especifica mediante un proceso de negocios usando una versión reducida y especializada de BPMN 2.0, para modelar la navegación entre páginas, la renderización de páginas, la ejecución de scripts, la conexión con servicios web y el intercambio de información entre servicios SMAPPS.

En la Figura 7-21 se muestra el diagrama de bloques para la creación de Servicios SMAPPS. El **proyecto BPMESMAPPS** define la configuración y utilitarios de un proyecto BPME, es decir, crear un subconjunto de elementos gráficos BPMN 2.0 para modelar un Servicio SMAPPS.

El funcionamiento del proyecto BPME está explicado en la sección anterior. Cabe recalcar aquí, que el resultado significativo del proyecto es la generación de un fichero XML que define el Servicio SMAPPS como proceso de negocio, este fichero es un programa Psi (definido en lenguaje BPMEPsi) que alimenta a la aplicación móvil, ver sección 7.5.4.

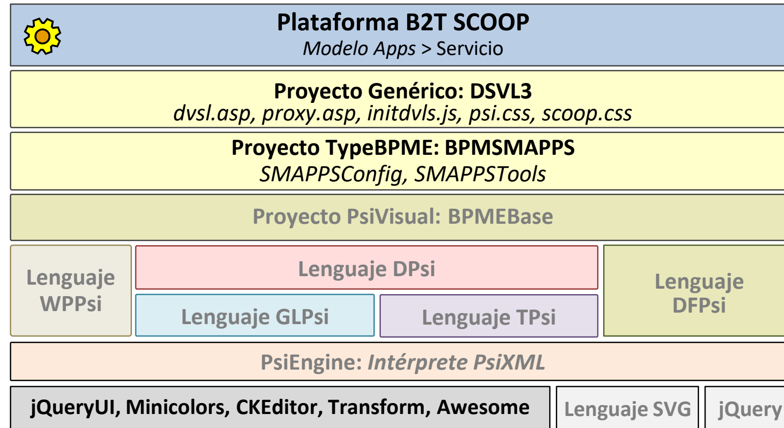


Figura 7-21. Diagrama de bloques para la creación de Servicios SMAPPS.

En la Tabla 7-4 se muestra la lista de proyectos Psi disponibles en el Proyecto SMAPPS. Los proyectos BPMEBase, DSVL3 y DFPsi, ya han sido mencionados en secciones anteriores.

Tabla 7-4. Proyectos Psi disponibles para el Proyecto Servicio SMAPPS

Proyecto	Tipo	Nombre	URL(*)
BPMEBase	D	Proyecto base BPME	http://hilas.ii.uam.es/project?d=BPMEBase
DSVL3	G	Interface con SCOOP	http://hilas.ii.uam.es/project?g=DSVL3
DFPsi	M	Lenguaje DFPsi	http://hilas.ii.uam.es/project?m=DFPsi
BPMSMAPPS	D	Configuración y utilitarios BPME para SMAPPS	http://hilas.ii.uam.es/project?d=BPMSMAPPS

Nota: Tipo: **G** (Proyecto Genérico); **M** (Proyecto PsiModel); **D** (Proyecto PsiDiagram). (*) También disponibles en CD adjunto.

7.5.2.1 Diagramas de Servicios SMAPPS

En la Figura 7-22 se muestra el editor de diagramas **BPMEEditor** (visualización del proyecto BPMEBase) es un DSVL para la definición de un Servicio SMAPPS. Como hemos visto en secciones anteriores, define cuatro tareas: (1) *Vista*, definición de página; (2) *Script*, definición de script en lenguaje JavaScript; (3) *Web Service,* definición de conexión a servicio web; y (4) *Almacenamiento*, almacenamiento de información la localStorage del WebPage en el móvil.

Adicionalmente, se tiene los elementos BPMN: *Gateway* (definición de Gateway exclusivo), *Evento Mensaje* (definición de mensaje de proceso), *Nodo Inicial*, *Enlace de Entrada*, *Enlace de Salida*, *Subproceso, Nodo de Terminación*, entre otros. Para cada elemento BPMN definido, en SCOPP se tiene las herramientas para la creación y asociación de contenido dentro del módulo *Modelos App* (Servicios de SCOOP).

En la Figura 7-22, cada elemento grafico BPMN del diagrama dispone de un menú flotante con las siguientes operaciones: (1) *configuración del elemento BPMN,* caja de diálogo especializada para configurar los elementos; (2) *descripción*, documentación del elemento; (3) *visor*, resumen de configuración del elemento por SCOOP y visualización del fichero *mapp.json*; (4) *consultar*, enlace para ir a la definición del elemento en SCOOP; y (5) *borrar,* borra el elemento BPMN).

En la Figura 7-22 se muestran además, los visores para todas las tareas (*vista*, *script*, *web service* y *almacenamiento*), el *gateway* y el *evento mensaje*. Además, se muestra el *menú principal* donde aparecen todas las opciones disponibles.

RESUMEN:

El editor de diagramas **BPMEEditor** es un DSVL en el cliente web que genera programas BPMEPsi y recursos para la creación de aplicaciones móviles basadas en Servicios SMAPPS de SCOOP.

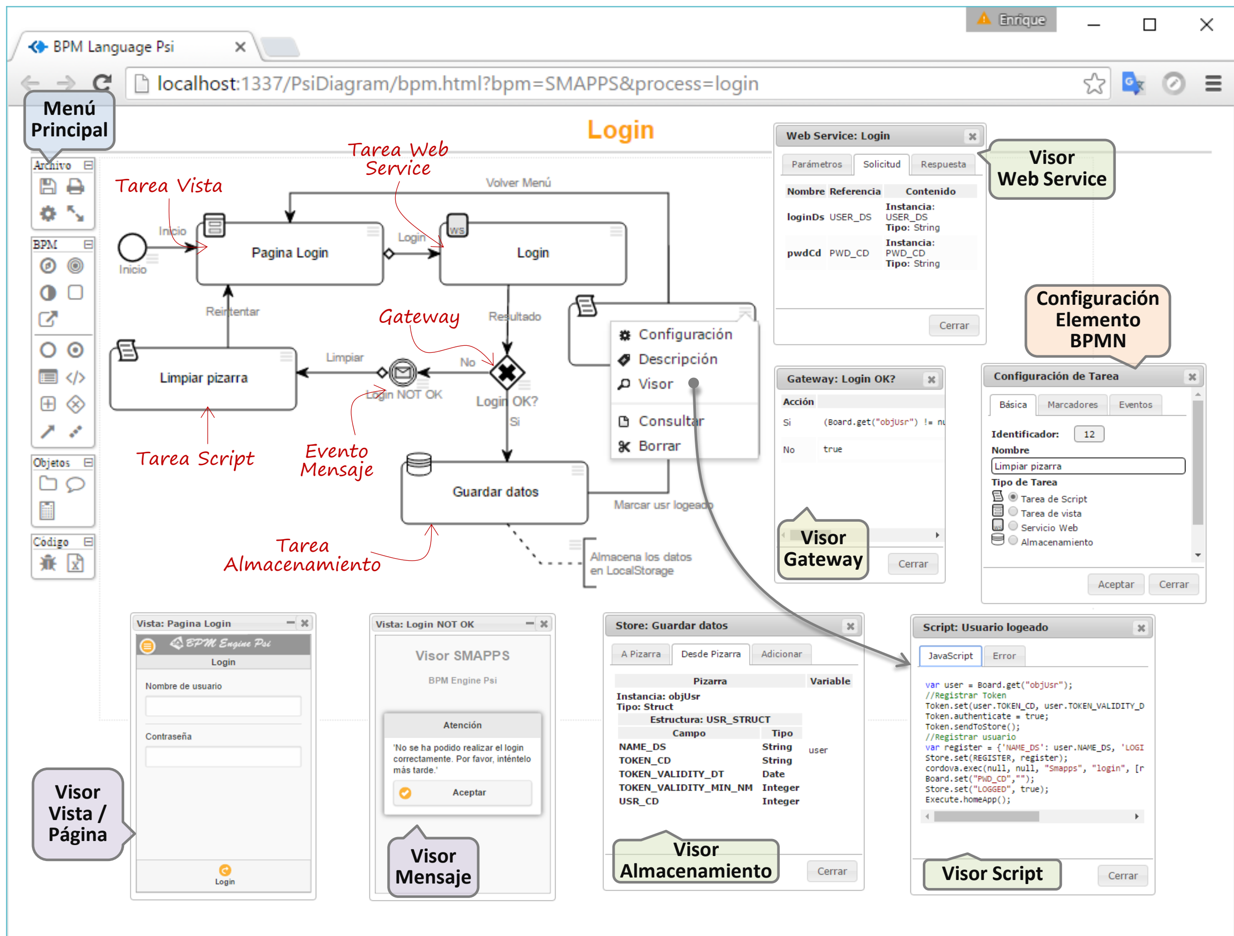


Figura 7-22. Editor de diagramas BPMEBase para la definición de Servicio SMAPPS de SCOOP.

7.5.2.2 Ejemplos de diagramas de BPME de la aplicación Móvil B2TConcept

Con el Visor de Proyectos BPMEs podemos acceder a los diagramas de BPME de la aplicación Móvil B2TConcept de la Figura 7-20. Los argumentos para la URL (7-1) son: *bpm*=SMMAPS & process=<process>. En la Tabla 7-5 se muestran URLs simplificadas para los diagramas de la aplicación Móvil B2TConcept.

Tabla 7-5. Diagramas BPME para la Aplicación Móvil B2TConcept

Diagrama	Descripción	URL
Login	Proceso de Login	http://hilas.ii.uam.es/visorSMAPPS?p=login
Gastos	Módulo de Gastos	http://hilas.ii.uam.es/visorSMAPPS?p=spend
Agenda	Agenda y control de horas	http://hilas.ii.uam.es/visorSMAPPS?p=agend

Nota: (*) También disponibles en CD adjunto.

7.5.3 Aplicación Móvil basada en Servicios SMAPPS

Una **Aplicación Móvil SMAPPS** es un conjunto generado de Servicios SMAPPS. Está aplicación puede crearse con dos tecnologías diferentes: (1) framework jQuery Mobile (jquerymobile.com) como se observa en la Figura 7-23(a), o, con (2) framework AngularJS (angularjs.org) y Angular Material (material.angularjs.org) como se observa en la Figura 7-23(b).

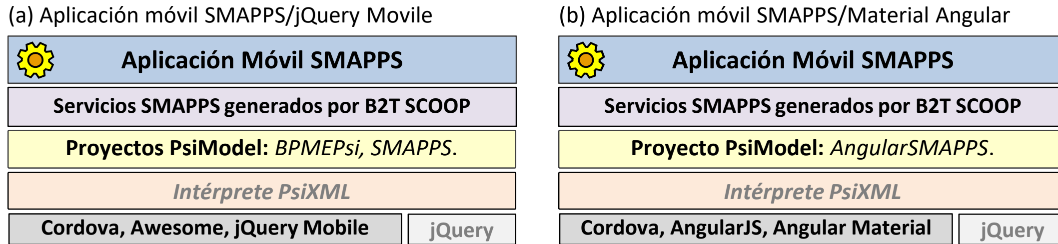


Figura 7-23. Diagrama de bloques de una Aplicación Móvil SMAPPS.

En la Tabla 7-4 se muestra la lista de proyectos Psi disponibles para las aplicaciones móviles SMAPPS.

Tabla 7-6. Proyectos Psi disponibles para las Aplicaciones Móviles SMAPPS

	Pr.	Proyecto	Nombre	URL(*)
jQuery Mobile		BPMEPsi	Motor de ejecución de BPM	http://hilas.ii.uam.es/project?m=BPMEPsi
	SMAPPS	Core	Kernel de ejecución de SMAPPS	http://hilas.ii.uam.es/project?m=SMAPPSCore
		Render	Renderizador de páginas basado en jQuery Mobile	http://hilas.ii.uam.es/project?m=SMAPPSRender
		Utils	Utilitarios	http://hilas.ii.uam.es/project?m=SMAPPSUtils
Material Angular	AngularSMAPPS	Core	Kernel de ejecución de SMAPPS	http://hilas.ii.uam.es/project?m=AngularSMAPPSCore
		Render	Renderizador de páginas basado en AngujarJS	http://hilas.ii.uam.es/project?m=AngularSMAPPSRender
		Run	Ejecución de aplicación BPM	http://hilas.ii.uam.es/project?m=AngularSMAPPSRun
		Utils	Utilitarios	http://hilas.ii.uam.es/project?m=AngularSMAPPSUtils

Nota: **Pr.**: Conjunto de proyectos; Todos los proyectos son de tipo PsiModel; (*) También disponibles en CD adjunto.

La primera versión de las aplicaciones móviles SMAPPS es el **proyecto BPMEPsi** (diciembre 2014). Este proyecto crea la primera versión del Lenguaje BPMEPsi y componentes adicionales para administrar sus recursos (fichero JSON), y se basa en tecnología jQuery Mobile.

Posteriormente, se realizó una refactorización y se crearon los **proyectos SMAPPS** (mayo 2015), tiene los mismos objetivos del proyecto BPMEPsi, pero se implementa en tres proyectos PsiModel diferentes: SMAPPSCore, SMAPPRender y SMAPPSUtils.

Con estas versiones se pusieron en funcionamiento tres productos: **Asisa** (manejo de seguros de clientes y ayuda en línea, descontinuada), **B2TConcept** (aplicación para el reporte de horas, gastos y agenda del B2TConcept, actualmente en producción) y Salas (prototipo para el manejo de recursos de salas y oficinas, en prototipo).

Con las nuevas tendencias y tecnologías basadas en AngularJS (angularjs.org, amplia el vocabulario HTML para crear vistas dinámicas para las aplicaciones web) y Material Design (material.io, diseño de materiales enfocado a la visualización del sistema operativo desarrollado por Google), surge *Angular Material*, que es un framework JavaScript de componentes UI bajo la especificación de Google's Material Design Specification (design.google.com). Se creó los proyectos AngularSMAPPS para aplicaciones móviles SMAPPS (agosto 2015 en pruebas) y mantiene la filosofía de crear una aplicación móvil SMAPPS basada en el Lenguaje BPMEPsi, pero se cambia la forma de renderizar sus páginas web. En la sección 7.5.4 se resumirá la implementación del Lenguaje Psi y los componentes para los proyectos AngularSMAPPS.

7.5.4 El Lenguaje BPMEPsi y los Componentes AngularSMAPPS

En esta sección se muestra el diseño e implementación del lenguaje BPMEPsi y los componentes para la renderización de páginas web en una aplicación móvil híbrida.

7.5.4.1 Lenguaje BPMEPsi: Componentes BPMEPsi e Implement

El **componente BPMEPsi** implementa el lenguaje BPMEPsi y tiene como objetivo ejecutar la funcionalidad de un Servicio SMAPPS. En la Figura 7-24 se muestra el PsiLSD del Lenguaje BPMEPsi y en el Anexo C.4 se muestra el PsiGVA para la gramática BPMEPsi.

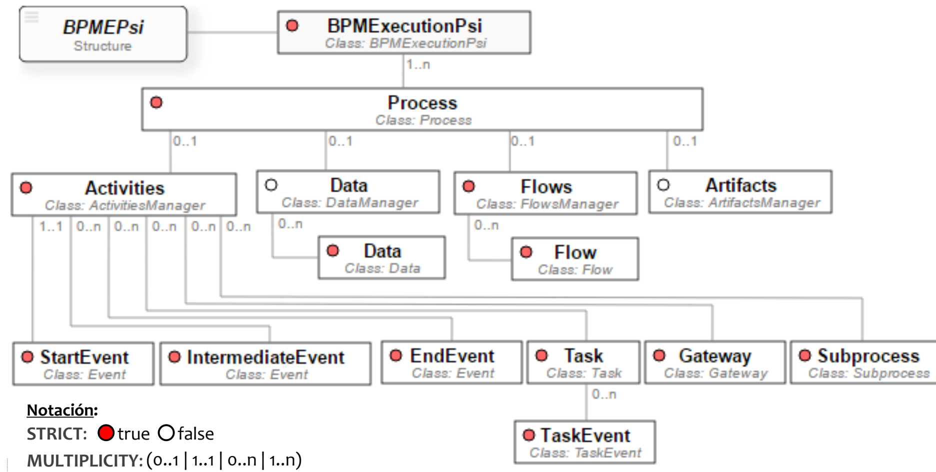


Figura 7-24. El PsiLSD del Lenguaje BPMEPsi.

En la Figura 7-25 se muestra el diagrama de clases para el componente BPMEPsi y su implementación para un servicio SMAPPS. El componente BPMEPsi implementa la lógica para ejecutar el flujo del servicio SMAPPS como se ha diseñado en un diagrama de Servicios SMAPPS (sección 7.5.2.1).

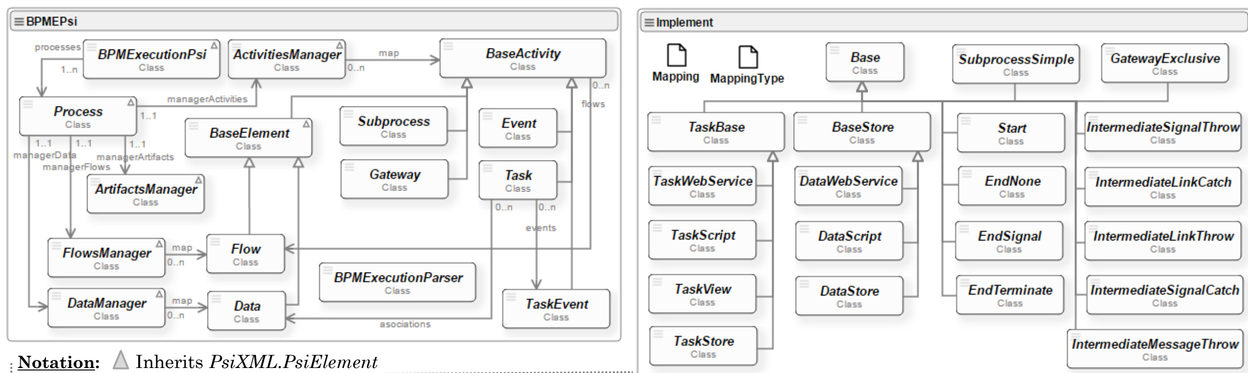


Figura 7-25. Diagrama de clases para el componente BPMEPsi y su implementación en un Servicio SMAPPS.

Cuando se ejecuta un Servicio SMAPPS, se crea una instancia de clase *Process,* se inicializa en el proceso todas las actividades, los datos y los flujos, y se crea la navegación abstracta del proceso. Al ejecutar el proceso, se evalúa cada elemento del proceso (flujo, tarea, compuerta, etc.) según la lógica definida, y para cada elemento, se crea una instancia de implementación (componente **Implement**) según corresponda. Por ejemplo, si se está ejecutando una tarea de vista, se crea una instancia de clase *TaskView* y este a su vez crea las instancias necesarias para renderizar una página web basado en la filosofía AngularJS y Material (implementadas en el componente **View**). Si se ejecuta la compuerta se

crea una instancia de la clase *GatewayExclusive* y se evalúa la expresión JavaScript con la información disponible en la pizarra del servicio SMAPPS. El proceso siempre tiene el control del elemento actual que se está ejecutando.

En la Tabla 7-7 se muestran las etiquetas y clases asociadas para el lenguaje BPMEPsi. El detalle de la implementación y la documentación no son disponibles por motivos de confidencialidad.

Tabla 7-7. Etiquetas y clases asociadas del lenguaje BPMEPsi

Etiqueta	Clase	Descripción de la clase
BPMEExecutionPsi* (1)	*BPMEExecutionPsi**	Punto de partida para la ejecución de un *Servicio SMAPPS* basada en la especificación de procesos de negocios BPMN 2.0.
Process (n)	*Process*	Lee el atributo url y vía AJAX se obtiene el documento XML, se compila y se evalúa con el Componente MIPsi, obteniendo la referencia a un objeto de la clase ModelImplementationPsi.
Activities (1)	*ActivitiesManager*	Administra la lista de actividades.
Flows (1)	*FlowsManager*	Administra la lista de flujos.
Artifacts (1)	*ArtifactsManager*	Administra la lista de artefactos.
Data (1)	*DataManager*	Administra la lista de datos.
StartEvent (1)	*Event*	Define un evento de inicio. Puede ser de tipo *simple* (inicio normal) o *señal* (llamado desde otro proceso) definidos en BPMN.
IntermediateEvent (n)	*Event*	Define un evento intermedio. Pueden ser de tipo *señal* (transferencia del flujo a otro proceso), *enlace* (enlace del flujo de dos puntos diferentes dentro del mismo proceso) y *mensajes* (representa una caja de diálogo modal que puede contener texto y/o controles visuales).
EndEvent (n)	*Event*	Define un evento final. Puede ser de tipo *terminación normal* (servicio terminado correctamente), *terminación con error* (terminación de un servicio con problemas en su flujo) y *terminación con señal* (termina el proceso y envía una señal para activar el siguiente proceso).
Task (n)	*Task*	Define un tipo de tarea: *vista* (tarea para renderización de una página web), *servicio web* (tarea para la conexión a un servicio web), *script* (tarea para la ejecución de un script en lenguaje JavaScript) y *almacenamiento* (tarea para el almacenamiento local). Cada una de ellas está especificada como elemento JSON.
TaskEvent (n)	*TaskEvent*	Manejo de los eventos de una tarea, varían según su tipo.
Gateway (n)	*Gateway*	Define una compuerta exclusiva (una sola salida para el flujo) de BPME 2.0. Su decisión se establece mediante comparaciones lógicas especificadas como expresiones JavaScript.
Subprocess (n)	*Subprocess*	El subproceso implementa el llamado a otro proceso. Este nuevo proceso pasa a formar parte del servicio, y cuando éste acaba devuelve el control del flujo al proceso padre.

Nota: *****: etiqueta o clase raíz; (1) Simple; (n) Múltiple.

7.5.4.2 Componentes View y Controls

En el proyecto AngularSMAPPSRender se definen los componentes **View** y **Controls**, y tiene como objetivo renderizar una página web y los mensajes de un Servicio SMAPPS. En la Figura 7-26 se muestra el diagrama de clases para el componente View (encargado de la creación de mensajes y páginas web), el componente Controls (lista de controles disponibles en las páginas web y en los mensajes) y el objeto *Templates* (define las plantillas y su uso de los diferentes controles).

Una página web (clase *Page*) contiene una lista de regiones (clase *Region*) y puede modificar la cabecera (objeto *Header*) y el pie de página (objeto *Footer*) según corresponda. Cada región contiene un conjunto de controles (clase *Control*) y son registrados en la página. Un control tiene una sola definición

de control (atributo *def*), y se pueden tener los siguientes tipos de definición: lista de radios (clase *RadioSetControl*), chequeo (clase *CheckboxControlDef*), lista de chequeo (clase *CheckSetControlDef*), interruptor (clase *SwitchControlDef*), lista de elementos (clase *ListControlDef*), área de texto (clase *TexareaControlDef*), títulos (clase *TilesControlDef*), matriz de iconos (clase *IconGridControlDef*), foto (clase *PhotoControlDef* y el objeto *Photo*), fragmento html (clase *HtmlControlDef*), imagen (clase *ImageControlDef*), lista de selección (clase *SelectControlDef*), botón (clase *ButtonControlDef*) y conjunto de botones (clase *ButtonSetControl*).

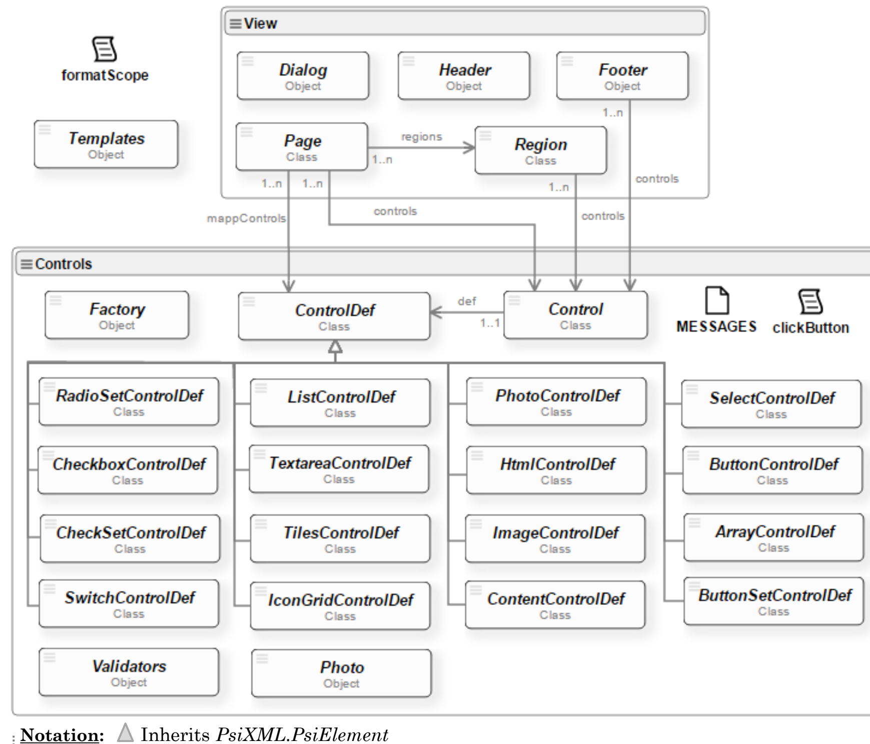


Figura 7-26. Diagrama de clases para renderizar una página web de un servicio SMAPPS.

7.6 RESUMEN

A lo largo del presente capítulo se ha mostrado la transferencia tecnológica vertida por las diferentes propuestas mostradas en la presente tesis a la empresa B2T Concept. Así, la propuesta de un modelo de programación (O2) y de un diagrama programable (O3) en un CSWE aporta a la Plataforma B2T (SCOOP) cuatro tipos de proyectos: Proyecto ADG (Authomatic Diagram Generator), Proyecto SID (SCOOP Interface Diagram), Proyecto BPME (Business Process Management Engine) y Proyecto SMAPPS (SCOOP Mobile Applications Service). Estos proyectos fueron desarrollados íntegramente en la aplicación web PsiEnvironment del capítulo anterior.

El Proyecto ADG permitió resolver los problemas de visualización de diagramas y de información a partir de fuentes heterogéneas en SCOOP. Entre los tipos de diagramas implementados están: diagrama de GANT, diagrama de PERT, conceptos y relaciones, diagrama de estados, modelo entidad-relación, llamado de funciones, mapa de objetos y diagrama WBS.

El Proyecto SID es una evolución del Proyecto ADG y cubre la sincronización entre la edición de diagramas y la información preexistente en la plataforma SCOOP. Define primitivas en SCOOP para la creación y edición de cualquier tipo de diagrama, basado en PsiDiagram. Entre los tipos de diagramas

implementados están: modelo de conceptos-relaciones, y, modelo de estados para el *Modelo Técnico*; modelo de conceptos-relaciones, modelo de estados y modelos de procesos para *Sketch*; y BPM para *BPMs y Eventos*.

El Proyecto BPME implementa la creación y edición de procesos de negocios basado en la especificación BPMN 2.0, para ser usado por SCOOP, aplicación web y/o widget. Tiene el Visor de Proyectos BPMEs, que es una página web que tiene como objetivo el de implementar proyectos de tipo BPME (por ejemplo, DSVL de Servicios SMAPPS).

Por último, el Proyecto SMAPPS establece un conjunto de componentes y un lenguaje Psi que dan soporte a aplicaciones móviles basados en Servicios SMAPPS de B2T SCOOP. Un Servicio SMAPPS modela la navegación entre páginas, la renderización de páginas, la ejecución de script, la conexión con servicios web y el intercambio de información. El editor de diagramas BPMEEditor es un DSVL en el cliente web que genera programas en lenguaje BPMEPsi (lenguaje para ejecutar la funcionalidad de un Servicio SMAPPS) y recursos para la creación de aplicaciones móviles basadas en Servicios SMAPPS de SCOOP. En otras palabras, el Proyecto SMAPPS consiste en generar el código de una aplicación móvil a partir de un DSVL de Servicios SMAPPS y fuentes heterogéneas de datos de SCOOP.

Capítulo 8. METODOLOGÍA DE VALIDACIÓN Y RESULTADOS

En este capítulo se presenta la metodología de validación, los casos de estudio y los resultados obtenidos que validan los objetivos de investigación planteados en esta tesis.

El primer paso a seguir, es resumir la metodología de validación (sección 8.1) y plantear una lista de características para validar el objetivo de investigación. En los capítulos anteriores se ha desarrollado la infraestructura necesaria para la creación de un modelo de programación y modelos gráficos en un CSWE sobre SCOOP. Así, en el Capítulo 3 se ha construído el motor de evaluación (PsiEngine) y una metodología de diseño e implementación para la creación y evaluación de XML-DSLs (denominados Lenguajes Psi) en un CSWE (O1). En el Capítulo 4 se diseñó y desarrolló un modelo de programación (PsiModel) para facilitar la creación de componentes JavaScript y componentes Psi (O2). En el Capítulo 5 se propuso el concepto de diagrama programable (PsiDiagram), para la creación de modelos gráficos y DSVLs en un CSWE (O3), y se dotó a PsiDiagram con la funcionalidad de generar y recomendar diagramas (O4). En el Capítulo 6 se implementó PsiEnvironment, basado en PsiModel y PsiDiagram, un ambiente de desarrollo ligero para facilitar la creación de lenguajes Psi, diagramas y DSVLs.

Este capítulo está dedicado a validar metodológicamente las características y/o funcionalidades propuestas en PsiEngine, PsiModel y PsiDiagram. Se verá cómo los distintos objetivos de investigación se cubren de acuerdo a los resultados obtenidos. Hay que destacar que la mayoría de los casos de estudio presentados en este capítulo, son proyectos que actualmente están en producción en sobre la plataforma SCOOP o en aplicaciones móviles.

8.1 METODOLOGÍA DE VALIDACIÓN

La investigación se ha realizado conforme a una **metodología cualitativa de casos de estudio** sugerida por (Yin, 2014) y adaptada para la Ingeniería de Software en (Baxter & Jack, 2008). En un CSWE y haciendo uso de PsiEngine, un caso de estudio se traduce en crear uno o varios componentes y/o lenguajes Psi, crear un conjunto de programas escritos en estos lenguajes Psi y tener *recursos* para obtener resultados o solucionar problemas específicos. Un resultado, por ejemplo, es crear componentes, widgets, XML-DSLs, aplicaciones web, diagramas, DSVLs, XML-DSLs y DSLJSONs.

La metodología citada en (Yin, 2014) se corresponde con el **tipo multicaso**. Con los casos de estudio propuestos se cubrirán los aspectos relevantes de cada uno de los objetivos de investigación. El tipo multicaso en este contexto se puede expresar como el conjunto de características y/o

funcionalidades (o simplemente características) a validar para cada objetivo de investigación. Cada caso cubre una parte de este conjunto, y el total de casos cubrirá el total de las características. Una característica puede ser validada por más de un caso de estudio. En general, aunque puede ser extremadamente costosa en tiempo y ejecución (Baxter & Jack, 2008), la evidencia creada a partir del multicaso se considera robusta y fiable.

Hay que destacar que un caso de estudio en el contexto de ésta tesis está relacionado con proyectos PsiModel o PsiDiagram creados y generados desde el PsiEnvironment, y estos son evaluados sobre diversas aplicaciones web y dispositivos.

Para acometer la validación, se requiere descomponer el objetivo general de la tesis en un conjunto de características a validar. Retomando nuevamente el objetivo general:

> La especificación, creación y evaluación de XML-DSLs con capacidad de asociar información heterogénea (XML/JSON) en un cliente web, para la construcción de componentes, widgets, XML-DSLs, DSVLs, modelos gráficos y diagramas que ayuden a dar soluciones a problemas específicos.

En la Tabla 8-1 se resumen diez características relevantes para el objetivo general de esta tesis que se denominará Validación del Objetivo General (**VOG**). Estás características se clasifican en tres grupos: PsiEngine, PsiModel y PsiDiagram. Además, en la columna "O.E." se muestra(n) lo(s) objetivo(s) específico(s) asociado a cada una de ellas.

Tabla 8-1. Validación del Objetivo General VOG: Resumen de características/funcionalidades a validar en el objetivo general de esta tesis agrupados en PsiEngine, PsiModel y PsiDiagram.

	Nº	Características	O. E.
PsiEngine	1	Ejecución de programas Psi: (S) simple-simple (un programa de un lenguaje Psi); (P) múltiple-simple (múltiples programas de un lenguaje Psi); (M) múltiple-múltiple (múltiples programas escritos en múltiples lenguajes Psi)	O1
	2	Asociación de información heterogénea: (X) XML; (J) JSON	O1
	3	Compartir objetos e información entre: (S) diferentes programas escritos en el mismo lenguaje Psi; (M) diferentes programas escritos en diferentes lenguajes Psi	O1
	4	Generación dinámica de una aplicación web por un programa Psi	O1
PsiModel	5	Creación de elementos de programación usados en PsiModel: (V) variable; (F) función; (E) tipo enumerado; (O) Objeto; (C) Clase; (W) Web widget	O2
	6	Construcción de componentes usados en PsiModel: (C) crear componente JavaScript; (P) crear componentes Psi (es decir, crear lenguajes XML-DSLs)	O2
PsiDiagram	7	Creación de diagramas basados en GLPsi: (S) figura; (L): línea; (C) contenedor; (A) lista; (K) comandos; (F) funciones; (E) Skeleton; (M) modificador gráfico	O3
	8	Creación de herramientas asociadas a diagramas con TPsi. *Herramientas Visuales* HV: (D) caja de diálogo; (B) barra de herramientas; (T) ventana emergente; (F) marco de contenido; (G) arrastrar / soltar; (R) soltar; (M) menú; (P) menú flotante. *Utilidades de Programación* PU: (V) variable; (F) función; (M) modificador gráfico; (C) Clases; (K) Comando Psi; (L) carga de documento	O3
	9	Creación de DSVLs basados en PsiDiagram	O3
	10	(G) Generación automática de diagramas basados en PsiDiagram, y, (R) creación de mecanismos de recomendación de diagramas basados en la generación automática de PsiDiagram	O3, O4

A modo de ejemplo ilustrativo, en la Tabla 8-2 se muestra un caso denominado CE que tiene por objetivo validar parte de las características de PsiEngine y PsiModel. El resultado del caso de acuerdo con la Tabla 8-1 es: se ha validado la asociación de información XML y JSON, ha validado que comparten

objetos e información de programas de un mismo lenguaje Psi. Además, CE ha creado un componente Psi que incluye funciones, clases y widgets y su objetivo fue generar una aplicación web.

Tabla 8-2. Ejemplo de validación de características del objetivo general.

Nº	Características	CE
2	Asociación de información heterogénea	XJ
3	Compartir objetos e información	S
4	Generación dinámica de una aplicación web por un programa Psi	☑
5	Creación de elementos de programación usados en PsiModel	FCW
6	Construcción de componentes usados en PsiModel	P

La VOG se dividirá en dos partes: (1) *validación de objetivos PsiModel y PsiEngine* (sección 8.2), requiere validar las características del 1 al 6 de la Tabla 8-1; y la (2) *validación de objetivos de PsiDiagram* (sección 8.3), requiere validar las características del 7 al 10 de la Tabla 8-1. Por último, en la sección 8.4 se hará un resumen y un consolidado de las validaciones de las características del objetivo general de esta tesis.

8.2 VALIDACIÓN DE OBJETIVOS PSIENGINE Y PSIMODEL

Para validar las características de PsiEngine y PsiModel del objetivo general de esta tesis, en la Tabla 8-3 se muestra un resumen de la lista de proyectos PsiModel desarrollados en PsiEnvironment, que nos sirven como casos de estudio. Los proyectos se han clasificado en cuatro grupos:

i. **Casos de Estudio Básicos (CB)**: los proyectos FeedPsi, MiniPsi, Anisha y Grapher están encaminados a validar la *totalidad de las características* de PsiEngine y PsiModel. En la sección 8.2.1, se describirán, analizarán e implementarán los casos de estudio y se realizará sus respectivos análisis de métricas y resumen VOG.

ii. **Kernel (KE)**: el proyecto PsiEngine implementa el PsiEngine y su Intérprete PsiXML y el proyecto MPsi implementa el modelo de programación PsiModel. Revalidan algunas características de PsiEngine y PsiModel, secciones 8.2.2 y 8.2.3.

iii. **Componentes Visuales (CV)**: los proyectos GLPsi, DPsi, TPsi, WPPsi y DFPsi implementan las funcionalidades para la creación de diagramas, modelos gráficos y DSVLs basados en PsiDiagram. Revalidan algunas características de PsiEngine y PsiModel, sección 8.2.4.

iv. **Componentes Móviles (CM)**: los proyectos BPMEPsi, SMAPPS jQuery y SMAPPS AngularJS para la creación de aplicaciones móviles hibridas usando BPME. Nuevamente revalidan algunas características de PsiEngine y PsiModel, sección 8.2.5.

Como se puede ver en la Tabla 8-3, actualmente en PsiEnvironment se tiene un total de 20 proyectos de componentes[10], de los cuales el 25% (5 proyectos) son casos de estudios y el 75% (15 proyectos) son componentes que están en producción.

De los proyectos en producción, en SCOOP están funcionando 7 componentes, mientras que en aplicaciones móviles hibridas funcionan 4 proyectos.

En todos los proyectos se han creado 35 componentes, de los cuales el 46% (16 proyectos) son componentes Psi, es decir, que se han implementado 16 lenguajes Psi en 13 proyectos. Además, se han

[10] En términos generales, los proyectos de componentes involucran tanto a componentes JavaScript, como componentes Psi.

creado 11 diagramas de clases basados en PsiDiagram que han sido presentados en diferentes apartados de esta tesis.

Tabla 8-3. Lista de proyectos PsiModel para la creación de componentes y componentes Psi.

Clasificación	Proyecto	TST	PRD	SCOOP	AM	CP	CPsi	DPsi	MPsi		MIPsi		Total		JS Gen.	Psi/JS
									#F	#L	#F	#L	#F	#L	#L	
Casos de Estudio Básicos (CB)	FeedPsi	☑				0	1	☑	1	54	1	160	2	214	490	2,3
	MiniPsi	☑				0	1	☑	1	60	1	192	2	252	519	2,1
	Anisha	☑				0	2	☑	1	60	1	141	2	201	494	2,5
	Grapher	☑				1	2	☑	1	230	4	1.131	5	1.361	2.145	1,6
	UPsi	☑				0	0		1	20	2	76	3	96	125	1,3
Kernel (KE)	**PsiEngine**		☑	☑	☑	3	0	☑	1	185	6	1.313	7	1.498	2.027	1,4
	PsiModel		☑			0	2	☑	1	341	6	825	7	1.166	2.339	2,0
Componentes Visuales (CV)	GLPsi		☑	☑		4	1	☑	1	407	7	2.497	8	2.904	4.226	1,5
	DPsi		☑	☑		2	1	☑	1	339	5	1.585	6	1.924	3.014	1,6
	TPsi		☑	☑		2	1	☑	1	464	9	2.298	10	2.762	4.201	1,5
	WPPsi		☑	☑		0	1	☑	1	129	5	684	6	813	1.028	1,3
	DFPsi		☑	☑		0	1	☑	1	149	2	403	3	552	1.149	2,1
Componentes Móviles (CM)	BPMEPsi		☑			2	1		1	571	12	4.185	13	4.756	6.300	1,3
	SMAPPS jQuery – Core		☑		☑	1	1		1	220	11	831	12	1.051	1.839	1,7
	SMAPPS jQuery – Render		☑	☑	☑	1	0		1	267	7	2.399	8	2.666	3.232	1,2
	SMAPPS jQuery – Utils		☑		☑	0	0		1	93	3	961	4	1.054	1.240	1,2
	SMAPPS AngularJS – Core		☑		☑	1	1		1	190	2	628	3	818	1.592	1,9
	SMAPPS AngularJS – Render		☑		☑	2	0		1	184	7	1.312	8	1.496	1.967	1,3
	SMAPPS AngularJS – Run		☑		☑	0	0		1	60	2	384	3	444	485	1,1
	SMAPPS AngularJS – Utils		☑		☑	0	0		1	88	3	950	4	1.038	1.222	1,2
Total		**5**	**15**	**7**	**8**	**19**	**16**	**11**	**20**	**4.111**	**96**	**22.955**	**116**	**27.066**	**39.634**	**1,5**

Nota: **Proyecto**: Proyecto de Componentes (Psi); **TST**: Caso de estudio de validación; **PRD**: Casos de estudio en producción; **SCOOP**: Proyectos usados en SCOOP; **AM**: Proyectos usados en aplicaciones móviles; **CP**: Número de componentes JavaScript; **CPsi**: Número de componentes Psi (XML-DSLs); **MPsi/MIPsi**: Programas de PsiModel; **DPsi**: Diagramación Psi; **Total**: Total de ficheros del proyecto; **JSGen.**: Fichero JS generado por PsiModel; **#F**: Número de ficheros; **#L**: Número de líneas; **Psi/JS**: Concisión de código Psi vs JavaScript generado.

Para PsiModel, se han escrito un total de 116 ficheros, de los cuales el 17% (20 ficheros) son especificaciones en MPsi, mientras que el 83% (96 ficheros) son implementaciones MIPsi. Con estos ficheros se han creado 27.066 líneas de código, de las cuales aproximadamente el 15% (4.111 líneas) son de especificaciones Psi y el resto (22.955 líneas) son implementación Psi.

Se han generado un total de 39.634 líneas de código JavaScript y comentarios, es decir, una concisión promedio de 1,5 de las líneas escritas en PsiModel. Las concisiones más altas son para los casos de estudio básicos, pero hay que destacar que el proyecto PsiModel con una concisión de 2. Algo relevante a afirmar aquí, es que el promedio de concisión de los proyectos que generan componentes Psi (ver Figura 8-2(a)) es del 1,8, mientras que los proyectos que generan componentes JavaScript (ver Figura 8-2(b)) es del 1,2. Esto se debe a que se genera más código JavaScript para los lenguajes Psi, implicando más fiabilidad, homogeneidad y reusabilidad.

Cabe recordar que el proyecto PsiEngine es la piedra angular de todos los proyectos, y el proyecto de PsiModel el que facilita la construcción de estos proyectos, como los proyectos de validación de PsiDiagram. Cada uno de ellos tendrá el análisis de métricas y el resumen VOG.

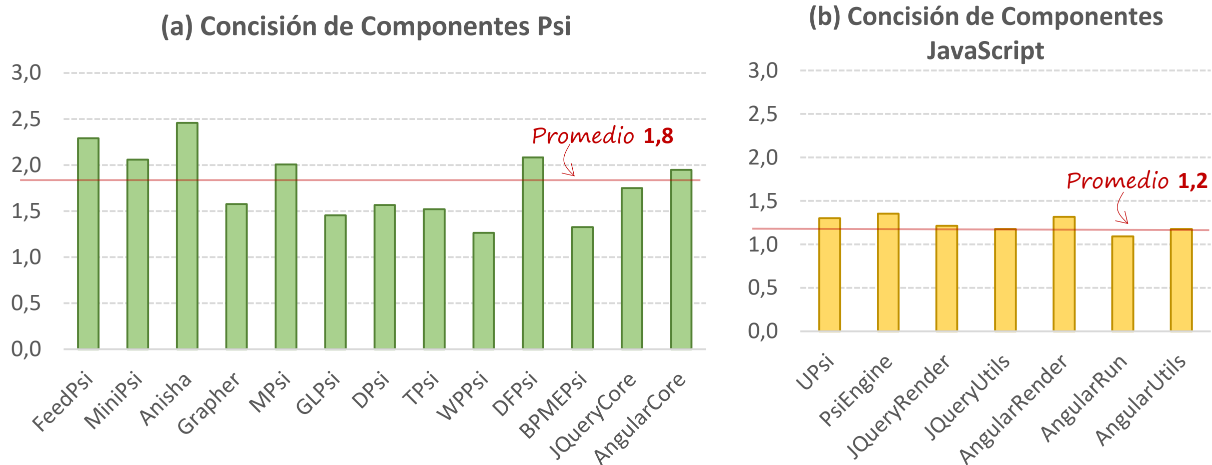


Figura 8-1. (a) Concisión de proyectos con Componentes Psi. (b) Concisión de proyectos con Componentes JavaScript.

8.2.1 Casos de Estudio Básicos

En esta sección se resumirán cuatros casos de estudio: *Aplicación Web FeedPsi*, *Lenguaje de Programación MiniPsi*, *Animación Anisha*, y *Diagramación Grapher*. El caso de estudio UPsi no se tendrá en cuenta para está validación.

Adicionalmente, se hará un análisis de métricas para los casos de estudio y un compendio del aporte a la validación de todas las características de PsiEngine y PsiModel.

8.2.1.1 Aplicación Web FeedPsi

Objetivo:

Construir y ejecutar una aplicación web de una página definida con el lenguaje FeedPsi. El lenguaje FeedPsi implementa la gestión y visualización de canales web basadas en RSS.

Un **canal web** es la fuente de difusión de contenidos (noticias, información, historia, novedades, tecnología, etc.) a través de la suscripción al canal. Con el objetivo de mantener al día de sus novedades a los usuarios inscritos. Con el uso de herramientas adecuadas (agregadores web como Netvibes, Feedly, Feed, Reader, Bloglines, etc.) los usuarios pueden mantener la información centralizada y clasificada. Existen diferentes formatos para la gestión y visualización de las canales web, siendo los más utilizados RSS (RSS Specifications, 2007) y Atom, ambos escritos con gramática XML.

El objetivo de este caso de estudio es ilustrar el uso de PsiEngine para soportar la ejecución de una aplicación web de difusión de noticias, a la que denominaremos **FeedPsi** (ver Figura 8-2). Para ello, se creó el Lenguaje FeedPsi y el Componente Feed. El análisis, diseño, especificación del lenguaje FeedPsi y el diagrama de clases se resumen en el Anexo E.1. Además, el detalle de la implementación del Componente Feed se muestra en http://hilas.ii.uam.es/feedpsi/api, o su código fuente disponible en el CD adjunto, o en http://github.com/echavarriaga/FeddPsi.

La aplicación **FeedPsi** gestiona las fuentes RSS permitiendo elegir, visualizar, almacenar por categorías y/o desechar noticias. Cada fuente RSS crea un submenú con los titulares de las noticias. Cuando se selecciona una noticia se muestra el resumen en el centro de la aplicación, con las opciones

de mostrar, ocultar y enviar a categoría. Una categoría es una lista de noticias guardadas para su posterior lectura.

En la Figura 8-2 se evalúa el programa “sport.xml”[11] escrito en FeedPsi. En general, con la URL:

http://hilas.ii.uam.es/FeedPsi/index.html?*list*=<*nombre*>

se pueden evaluar otros programas FeedPsi, donde <*nombre*> puede ser science, spain, sport-es, technology y journal.

Figura 8-2. Aplicación Web FeedPsi. Agregador de noticas al estilo Psi.

La aplicación FeedPsi, tiene la ventaja de no requerir para su programación el uso de ninguna tecnología del servidor, siendo posible efectuar todo el despliegue de la aplicación en el cliente web, con un mínimo coste computacional en el servidor.

8.2.1.2 Lenguaje de Programación Ligero MiniPsi

> **OBJETIVO**:
> Crear XML-DSL ligero para la definición de variables, funciones clases, instancias de clase y bloques de código JavaScript. Además Implementa el DAO para XML/JSON.

Este caso de estudio se presentó en la sección 3.7 como ejemplo ilustrativo de creación de un lenguaje Psi para el PsiEngine y el uso de la metodología propuesta en sección 3.4. Tiene como objetivo

[11] Programa Sports: http://hilas.ii.uam.es/FeedPsi/source/sports.xml

validar la ejecución de múltiples programas de un lenguaje Psi. En la Figura 3-12 se mostró la aplicación web que se ha desarrollado para el lenguaje MiniPsi.

En http://hilas.ii.uam.es/minipsi/examples se ilustran con ejemplos los elementos básicos del lenguaje MiniPsi y la asociación de información XML. Además, se pone a disposición del usuario un conjunto de clases MiniPsi para crear un entorno gráfico sencillo basado en el *canvas* de HTML. El código fuente de este caso está disponible en el CD adjunto, o en http://github.com/echavarriaga/FeddPsi.

8.2.1.3 Animación Anisha

OBJETIVO:

Construir y ejecutar una animación basada en formas básicas de SVG. Por un lado se crea el lenguaje ShapesPsi para crear círculos y rectángulos, y un lenguaje AnimePsi para animar estos elementos. Por otro lado, se crea una página web para ejecutar ejemplos.

El objetivo de Anisha es definir elementos de formas básicas (círculo y rectángulo) y animarlos sobre un canvas SVG. Se definen dos lenguajes Psi: **ShapesPsi** y **AnimePsi**. El ShapesPsi define círculos y rectángulos sobre un canvas SVG y los pone a disposición de otros programas Psi, y en particular a programas escritos en AnimePsi. El AnimePsi toma estos elementos y les proporciona animación.

Este caso de estudio tiene como objetivo principal ilustrar la construcción y coexistencia de múltiples lenguajes Psi y la evaluación de múltiples programas, permitiendo asociar información XML. El análisis, diseño, especificación de ShapesPsi y AnimePsi y el diagrama de clases se resumen en el Anexo E.2. Además, el detalle de la implementación de los componentes Shapes y Anime se muestran en http://hilas.ii.uam.es/anisha/api.

La página de ejemplos ilustrativos se encuentra en http://hilas.ii.uam.es/anisha. En la Figura 8-3 se muestra una captura del ejemplo “runs”. Se evalúa un programa ShapesPsi con dos elementos (Circle y Rectangle) a los cuales se asocia información desde un fichero XML y un objeto JavaScript. Por otro lado se evalúa un programa AnimePsi que mueve estos dos elementos. El código fuente de este caso de estudio está disponible en el CD adjunto, o en http://github.com/echavarriaga/Anisha.

8.2.1.4 Diagramación Grapher

OBJETIVO:

Crear diagramas basados en los elementos gráficos figura y línea. La definición de los elementos se registran en una librería gráfica y la diagramación se realiza en base a está librería sobre un lienzo SVG.

Este caso de estudio es el prototipo de PsiDiagram. Por un lado se define una librería gráfica SVG (figura y línea) con el lenguaje GraphsPsi y una diagramación con el lenguaje PaintPsi. Por otro lado, se implementa la aplicación web Grapher que es un entorno ligero para construir diagramas basados en los lenguajes GraphsPsi y PaintPsi.

En la Figura 8-4 se muestran las partes que intervienen en la creación y visualización de un diagrama a partir de dos elementos gráficos (línea y figura): la *librería gráfica*, las *fuentes de información* y la *diagramación*. La *librería gráfica* tiene como objetivo definir los elementos gráficos línea y figura basados SVG que sean manipulables, transformables y reutilizables. Las *fuentes de información* son ficheros de datos XML o JSON asociados al diagrama y median en su apariencia y contenido. Por último,

la *diagramación* es la representación gráfica del diagrama con elementos SVG a partir de la *librería gráfica* y las *fuentes de información*, ésta representación se hace directamente sobre un lienzo SVG.

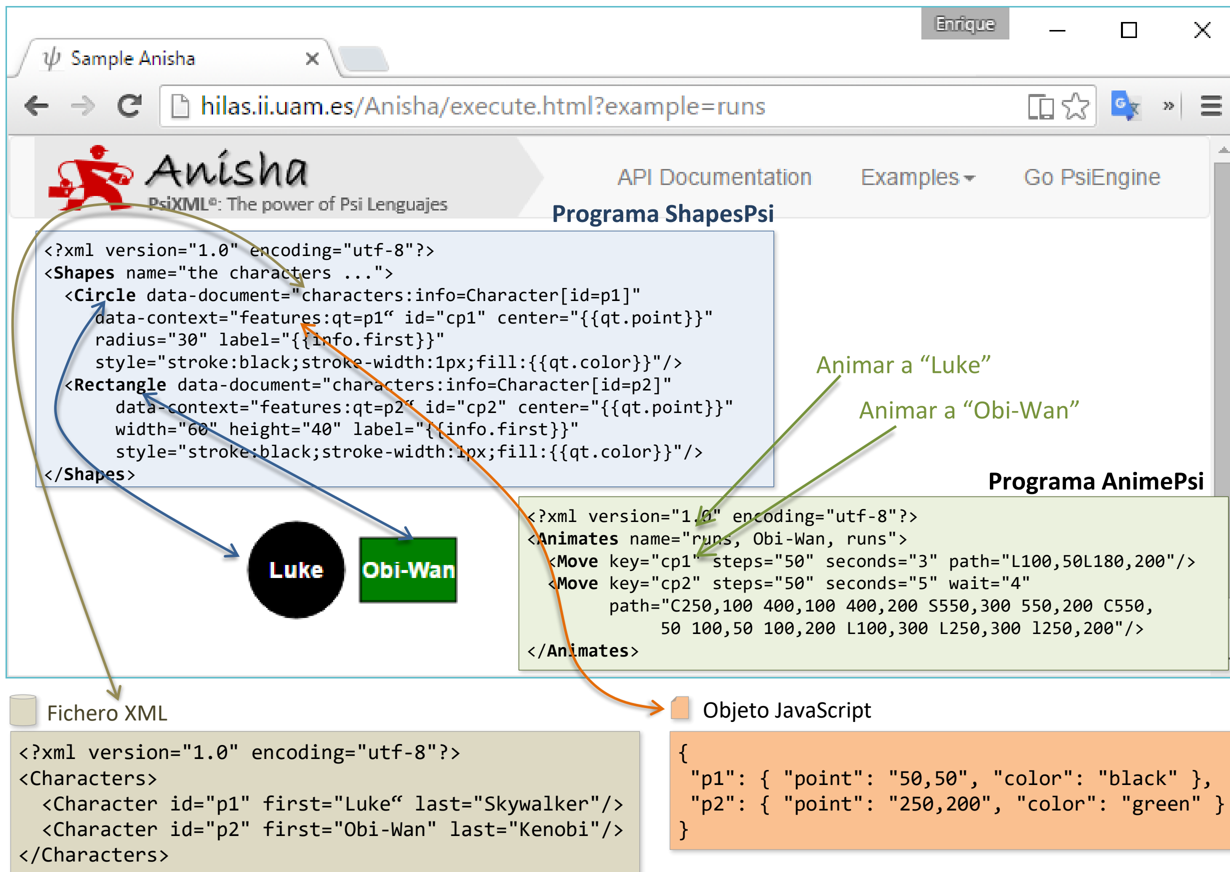


Figura 8-3. Ejemplo ilustrativo de Anisha.

En consecuencia, para crear y visualizar diagramas asociando diferentes fuentes de información, se plantean los componentes Psi: **Componente Graphs** y **Componente Paint**. El componente Graphs implementa el lenguaje **GraphsPsi**, este lenguaje administra los elementos gráficos línea y figura de la *librería gráfica*. El componente Paint implementa el lenguaje **PaintPsi** encargado de la *diagramación* a partir de *librerías gráficas*, con la posibilidad de asociar la información XML o JSON.

El objetivo de este caso de estudio es ilustrar cómo el uso de PsiEngine soporta la construcción de múltiples lenguajes Psi y la ejecución de múltiples programas, con la capacidad de asociar información heterogénea XML/JSON. El análisis, diseño, especificación de GraphsPsi y PaintPsi y el diagrama de clases se resumen en el Anexo E.3. Además, el detalle de la implementación de los componentes Graphs y Paint se muestran en http://hilas.ii.uam.es/grapher/api.

Una vez implementados los componentes reutilizables Graphs y Paint se implementa la **Aplicación Web Grapher**, que es un entorno de desarrollo ligero para la creación de diagramas y modelos gráficos basados en proyectos, en la edición de ficheros XML y en el uso de los componentes Graphs y Paint (http://hilas.ii.uam.es/grapher). El código fuente de este caso de estudio está disponible en el CD adjunto, o en http://github.com/echavarriaga/Grapher.

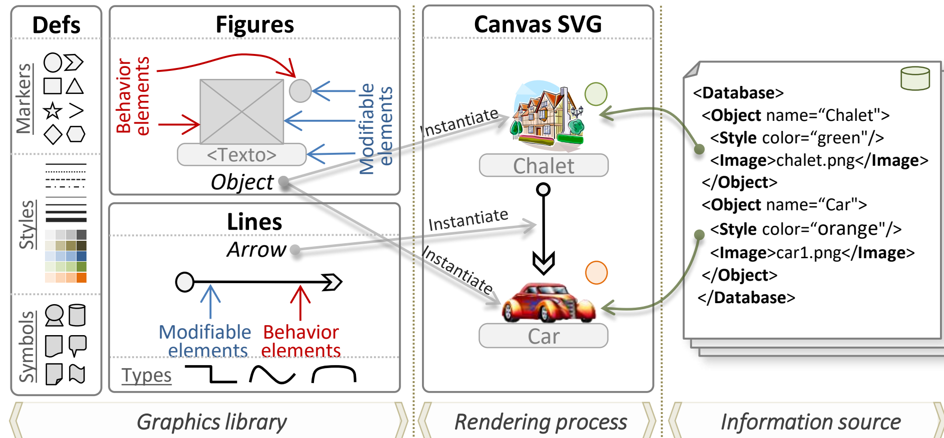


Figura 8-4. Creación y visualización de figuras y líneas basadas en el concepto de definición gráfica.

En la Figura 8-5, Grapher se divide en dos secciones: *Proyecto* y *Diagrama*. Un *Proyecto* es un conjunto de ficheros de librerías gráficas (ficheros GraphsPsi), diagramación (ficheros PaintPsi) y fuentes de información XML/JSON (ficheros XML/JSON). El *Diagrama* es un canvas SVG donde se evaluarán los programas de diagramación.

Los editores GraphsPsi y PaintPsi permiten editar ficheros de código fuente GraphsPsi y PaintPsi, respectivamente del proyecto activo. Adicionalmente, se implementa editores para ficheros de datos XML y JSON.

8.2.1.5 Análisis de métricas de proyectos

Una vez presentados los cuatro casos de estudio, está subsección está dedicada a analizar las métricas software para los componentes implementados, validando el objetivo general de esta tesis respecto a las características de PsiEngine y PsiModel.

Para cada caso de estudio se crearon dos tipos de proyectos en PsiEnvironment. Por un lado, el proyecto para crear el (los) componente(s) Psi respectivo(s) (proyecto PsiModel) y, por otro lado, una aplicación web que usa y verifica estos componentes Psi (proyecto Genérico).

En la Tabla 8-4 se muestra el resumen de ficheros que conforman los proyectos Genéricos de aplicación web para los cuatro Casos de Estudio Básicos. Se muestran el número de ficheros [X]Psi, HTML, CSS, XML, XSL, JSON y otros recursos (imágenes, textos, etc.). Donde [X]Psi es el número programas Psi creados para cada lenguaje Psi en cada caso de estudio. No se incluyen los ficheros de documentación y de ayuda.

Tabla 8-4. Lista de proyectos Genéricos de los Casos de Estudio Básicos en PsiEnvironment.

Caso de Estudio	Proyecto	[X]Psi	HTML	XML	JSON	CSS	JS	XSL	Otros	Total
Aplicación web	FeedPsi	5	1	0	0	2	4	4	20	36
Lenguaje de programación	MiniPsi	10	1	2	1	3	12	0	2	31
Animación	Anisha	2	1	1	1	2	5	0	4	16
Diagramación	Graphers	14	3	3	3	5	25	0	12	65
	Total	31	6	6	5	12	46	4	38	148

Nota: **Proyecto**: Nombre de proyecto; **[X]Psi**: Programas Psi; **Otros**: Imágenes, texto, SVG, etc.

En la Tabla 8-5 se lista la creación de los componentes Psi en PsiEnvironment para cada caso de estudio. En cada proyecto se puede calcular sus respectivas métricas software.

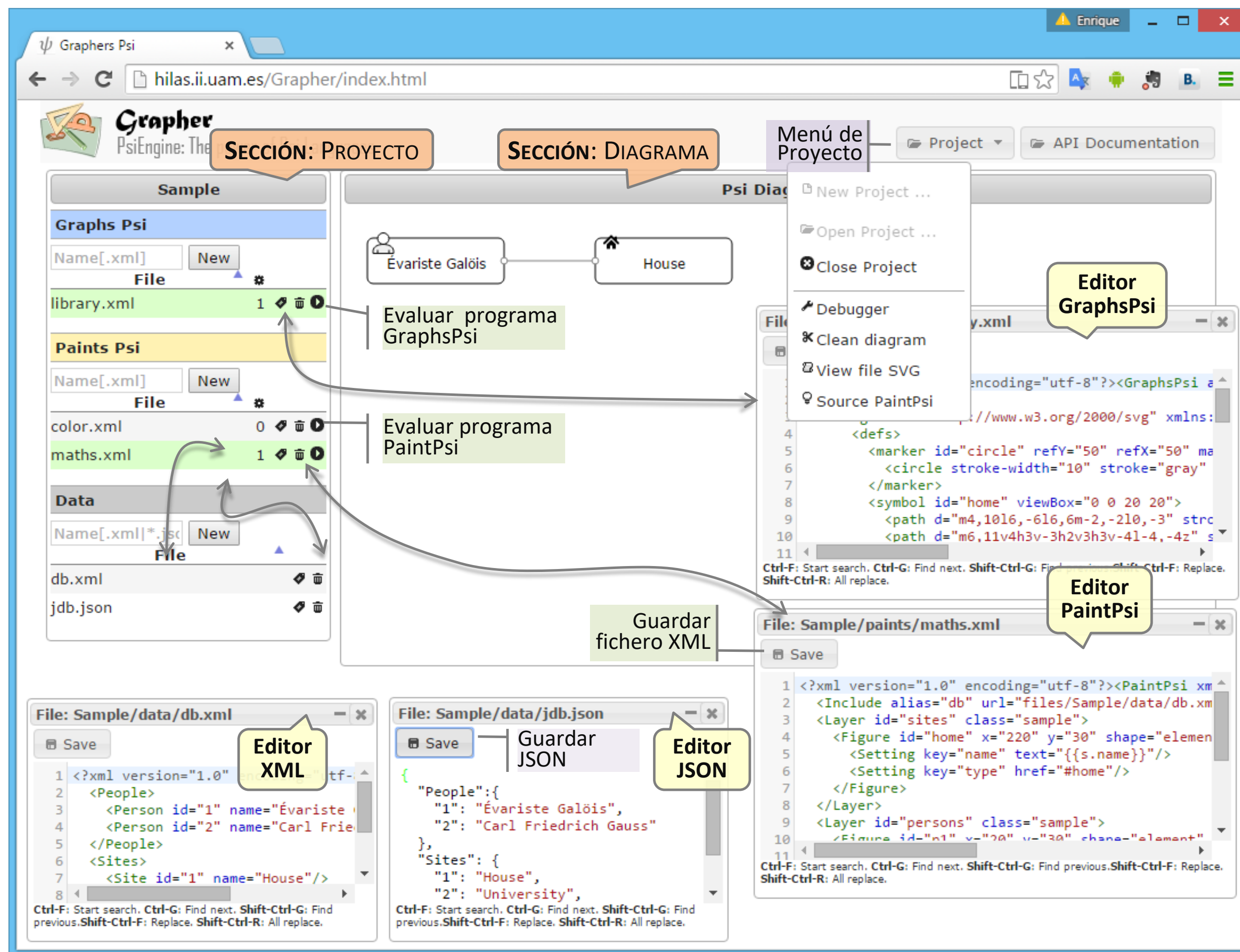


Figura 8-5. *Grapher*: entorno de desarrollo ligero para el uso de los componentes Graphs y Paint.

En los 4 proyectos de componentes, se crearon 11 ficheros PsiModel, de los cuales el 36% (4) son de especificación Psi y el 64% (7) son de implementación Psi. Se escribieron un total de 2.028 líneas de código PsiModel, con las que se generaron un total de 3.648 líneas de código JavaScript (es decir, una concisión promedio de 1,8). Además, todos los componentes disponen de diagramas de clases y diagramas PsiLSD basado en diagramas Psi.

Tabla 8-5. Proyectos FeedPsi, MiniPsi, Anisha y Grapher de tipo PsiModel en PsiEnvironment.

Caso de Estudio	MPsi	MIPsi	#M	#LC	Dg.	Fichero	#LG	#LG/#LC
Aplicación Web FeedPsi *URL*: http://hilas.ii.uam.es/project?m=FeedPsi	1	1	2	214	☑	FeedPsi.js	490	2,3
Lenguaje MiniPsi *URL*: http://hilas.ii.uam.es/project?m=MiniPsi	1	1	2	252	☑	MiniPsi.js	519	2,1
Animación Anisha *URL*: http://hilas.ii.uam.es/project?m=AnishaPsi	1	1	2	201	☑	Anisha.js	494	2,5
Diagramación Graphers *URL*: http://hilas.ii.uam.es/project?m=GrapherPsi	1	4	5	1.361	☑	Grapher.js	2.145	1,6
Total	**4**	**7**	**11**	**2.028**			**3.648**	**1,8**

Nota: **MPsi/MIPsi**: Programas de PsiModel; **Dg.**: Diagrama de Clases Psi; **#M**: Total MPsi+MIPsi; **#LC**: Total de líneas de Psi; **Fichero**: Fichero generado; **#LG**: Líneas generadas JavaScript. Código fuente también disponible en DC adjunto.

En la Figura 8-6 se resume algunos de los resultados obtenidos de las métricas para los cuatro Casos de Estudio Básicos. Para obtener la información completa de las métricas para cada caso de estudio, obtener puede obtener su resultado directamente en el proyecto.

En relación con el número de Elementos de Programación se desarrollaron un total de 59 elementos, de los cuales el 62% (37) son clases. Se crearon un total de lenguajes Psi implementados en sus respectivos componentes Psi (10%), mientras que el resto de elementos cubre el 28%. En particular, se cumple con el objetivo de tener al menos un elemento definido en alguno de los casos de estudio, para validar los elementos de programación de PsiModel.

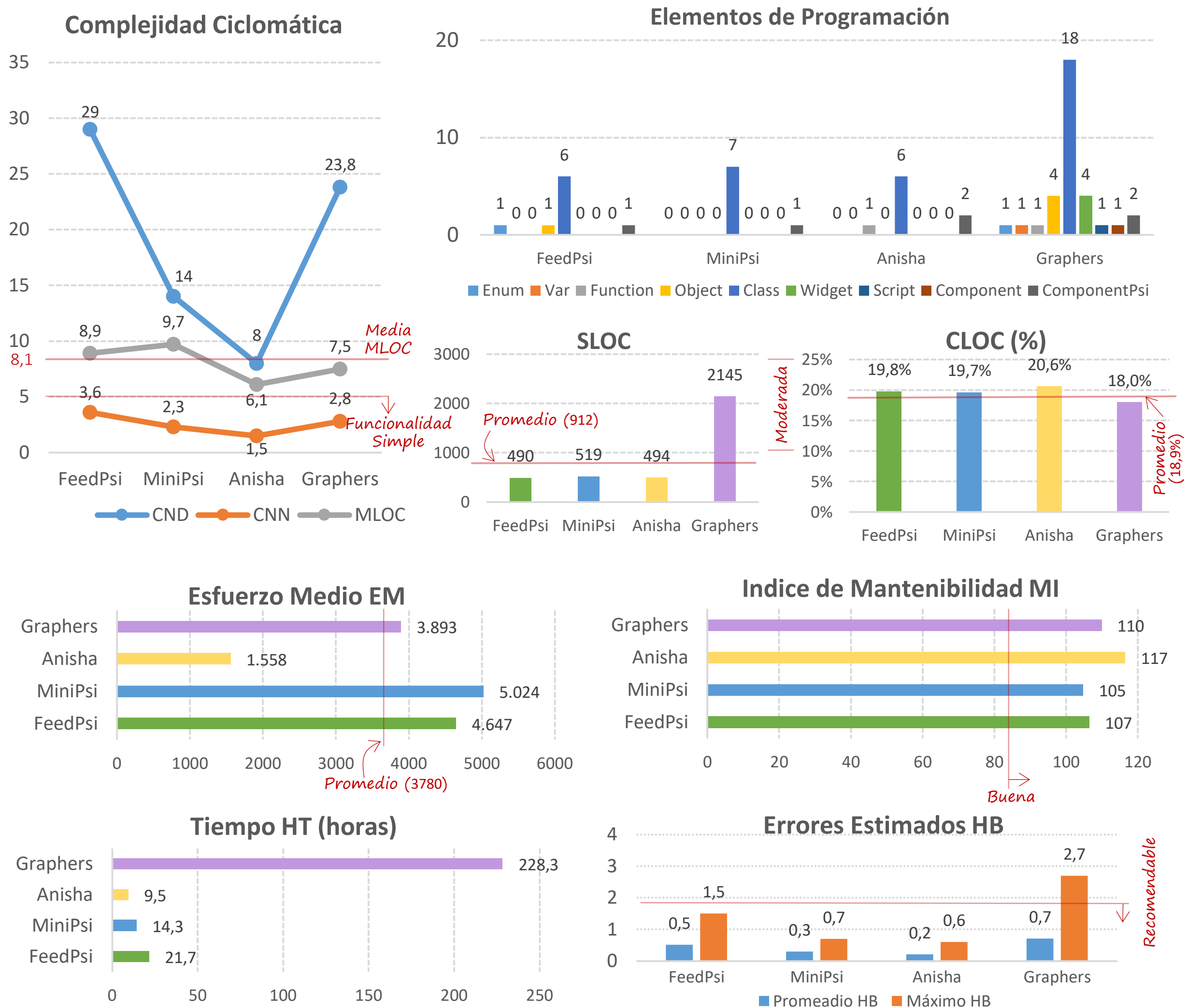


Figura 8-6. Resumen de métricas para los Casos de Estudio Básicos.

La complejidad ciclomática promedio CNN es baja para los cuatro casos, es decir, tienen una funcionalidad simple. Esta afirmación se puede ver reflejada en el número medio MLOC de 8,05 de todas las funciones existentes.

Por otro lado, el total SLOC es de 2535 líneas en todos los casos de estudio, de las cuales 18,9% son líneas comentarios y están en el rango moderado. Además, el 59% (2145 SLOC) son del componente Grapher. Cabe mencionar que este componente es base de implementación de la librería y la diagramación Psi de PsiDiagram. Además, el índice de mantenibilidad MI para los cuatro casos es

superior a 85, es decir, todos los componentes tienen buena capacidad de mantenimiento. Cabe destacar aquí que el MI de Grapher es mejor que FeedPsi y MiniPsi y se debe principalmente a su menor esfuerzo medio EM.

El tiempo HT de implementar o entender un programa para Grapher (228 horas) se debe a su longitud y vocabulario (11243 HN y 858 Hn) aproximadamente 6 veces mayor al promedio de longitud y vocabulario de los otros tres casos de estudio, o en otras palabras, al doble de la suma de longitud y vocabulario de los tres casos de estudio. En relación al número esperado de errores HB, para Graphers es bastante elevado (36,5).

En la Tabla 8-6 se muestran las métricas para los componentes y componentes Psi de los Casos de Estudio Básicos. En términos generales, se puede concluir como resumen que los cuatro casos de estudio tienen indicadores de software bastante adecuados, ya que sus índices de mantenibilidad MI son buenos, sus errores estimados HB son recomendables y con una documentación moderada.

Tabla 8-6. Métricas para los componentes y componentes Psi de los Casos de Estudio Básicos.

Componente	CE	T	CNA	CND	CNN	EM	MLOC	NOP	MI	HN	Hn	HV	HD	HE	HT
FeedPsi	F	P	103	28,9	3,6	4.647,4	8,9	0,6	106,5	2.451	252	19.552,3	71,2	1.391.301,4	21,5
MiniPsi	M	P	46	13,8	2,3	5.023,6	9,7	0,2	104,8	1.752	248	13.935,8	65,6	914.773,4	14,1
Shapes	A	P	12	8,3	1,4	1.365,3	5,3	0,2	119,2	791	155	5.755,4	36,1	207.541,1	3,2
Paint	A	P	8	7,4	1,4	1.817,3	6,6	0,3	114,6	545	137	3.868,4	30,1	116.290,4	1,8
GrapherApp	G	J	74	27,2	3,5	4.400,5	9,3	1,2	105,9	1.940	232	15.244,5	61,3	934.897,2	14,4
Graphs	G	P	72	20,6	2,3	3.484,1	6,6	1,1	112,4	2.605	285	21.243,3	73,3	1.556.748,5	24,0
Paint	G	P	144	24,9	3,4	5.775,2	9,6	0,4	104,4	3.923	371	33.483,9	90,1	3.015.312,0	46,5

Nota: **CE**: Caso de estudio, F=FeedPsi, M=MiniPsi, A=Anisha, y G=Graphers; **T**: Tipo de componente J (JavaScript), P (Psi).

8.2.1.6 Resumen VOG de los Casos de Estudio Básicos

En la Tabla 8-7 se muestra el resumen de características validadas para el proyecto PsiEngine y los casos de estudio aplicación web FeedPsi, lenguaje de programación MiniPsi, animación Anisha, y diagramación Graphers.

Cabe notar que los casos de estudio fueron completamente desarrollados en el entorno de desarrollo PsiEnvironment cubriendo la totalidad de las características para los objetivos de investigación O1 y O2. Además se resalta que el caso de estudio Grapher es la base para la implementación de los lenguajes GLPsi y DPsi del PsiDiagram.

Tabla 8-7. VOG: Resumen de características validadas en los Casos de Estudio Básicos para PsiEngine y PsiModel.

Nº	Características	Proyectos			
		FeedPsi	MiniPsi	Anisha	Grapher
1	Ejecución de programas Psi	S	P	M	M
2	Asociación de información heterogénea	X	X	XJ	XJ
3	Compartir objetos e información	☐	S	M	M
4	Generación dinámica de una aplicación web por un programa Psi	☑	☐	☐	☐
5	Creación de elementos de programación usados en PsiModel	EOC	C	FC	VFEOCW
6	Construcción de componentes usados en PsiModel	P	P	P	CP

CONCLUSIÓN:

Se han validado TODAS las características de PsiEngine y PsiModel de VOG, es decir características de 1-6 de la Tabla 8-1.

Como validaciones adicionales de las características mencionadas, se mostrarán el resto de casos de estudios que están representados por los proyectos: Proyecto PsiEngine, Proyecto PsiModel, Proyectos de PsiDiagrama y Proyectos Móviles.

8.2.2 KERNEL: PROYECTO PSIENGINE

En esta sección se resumirá el análisis de las métricas para PsiEngine y su Intérprete PsiXML. Además, se determina el aporte a la validación de las características para PsiEngine y PsiModel. En el proyecto PsiEngine de PsiEnvironment, se especifican e implementan las componentes del PsiEngine: *Intérprete PsiXML* (PsiXML), *Lenguajes Psi* (LanguagePsi) y *utilitarios de mensajes* (PsiText) descritos en el Capítulo 3. En la Tabla 8-8 se resume como está conformado este proyecto.

Tabla 8-8. Proyecto PsiEngine en PsiEnvironment.

Caso de Estudio	MPsi	MIPsi	#M	#LC	Dg.	Fichero	#LG	#LG/#LC
PsiEngine (PsiText, PsiXML, LanguagePsi) *URL*: http://hilas.ii.uam.es/project?m=PsiEngine	1	6	7	1.498	☑	PsiXML.js	2.027	1,4

Nota: **MPsi/MIPsi**: Programas de PsiModel; **Dg.**: Diagrama de Clases Psi; **#M**: Total MPsi+MIPsi; **#LC**: Total de líneas de Psi; **Fichero**: Fichero generado; **#LG**: Líneas generadas JavaScript. Código fuente también disponible en DC adjunto.

8.2.2.1 Análisis de métricas

En la Figura 8-7 se muestran las métricas del proyecto de componentes PsiEngine generadas de forma automática por PsiEnvironment.

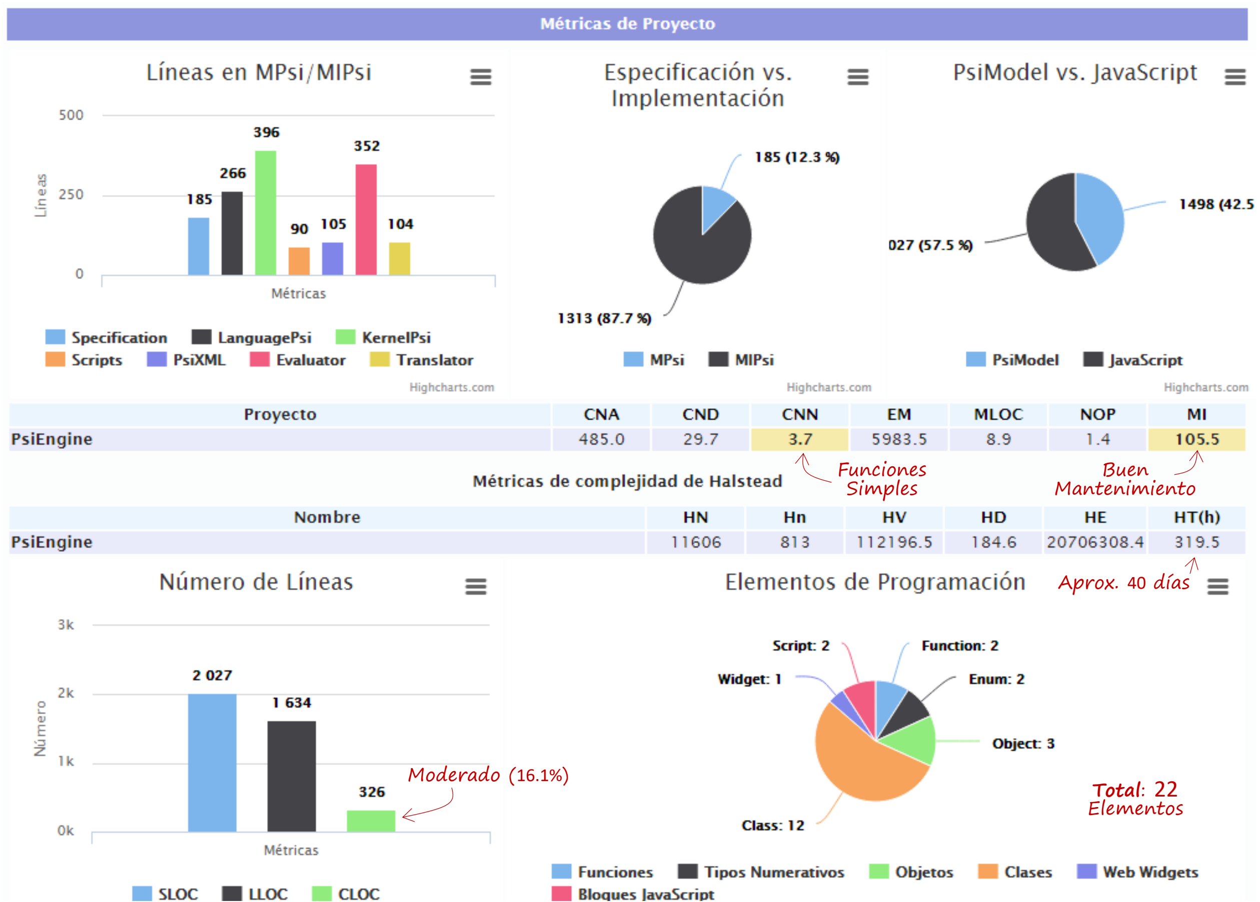


Figura 8-7. Resumen de métricas del proyecto de componentes PsiEngine.

Como puede apreciarse, la complejidad ciclomática promedio CNN de las funciones/métodos del proyecto son “funcionalidades simples”, apunta a una buena capacidad de mantenimiento MI y las líneas de comentario CLOC son moderadas (16,1%). Además, el tiempo estimado para implementar es de aproximadamente 40 días (39,3 días), mientras que el total de elementos de programación es 22 donde el 55% (12) son clases. Estos resultados se obtienen analizando el proyecto en conjunto.

Al analizar el compendio de métricas por elemento de programación, el mayor esfuerzo promedio EM se requiere en la creación de clases (41%), tiene el mayor tamaño de vocabulario (7.296), la mayor densidad ciclomática HD y se requieren aproximadamente 187,4 horas. Por otro lado, el número de parámetros promedio NOP y su mantenibilidad es buena para todos los elementos de programación.

En la Figura 8-8 se muestra parte de la tabla de la métrica de elementos generada por PsiEnvironment. Analizando los resultados, la clase abstracta *PsiElement* de PsiXML es la clase más compleja de los elementos de programación de PsiEngine. Este dato era predecible dado que es la piedra angular de los lenguajes Psi y tiene la capacidad de procesar los lenguajes PsiLI y PsiCA. Su CNN tiene una funcionalidad normal, refleja buen mantenimiento (pero es el valor más bajo de todos los elementos) y es la que requiere mayor esfuerzo para su implementación. La estimación del error sobrepasa el límite aconsejado, implicando que es necesario hacer una revisión de la clase.

En términos generales, el PsiEngine con las métricas obtenidas reflejan una complejidad normal y la estimación de posibles errores está dentro de los límites.

Métricas de Elementos

Nombre	tipo	CNA	CND	CNN	EM	MLOC	NOP	MI	HN	Hn	HV	HD	HE	HT(h)	HB
Scanner	Class	16.0	40.0	4.0	4122.9	6.8	1.2	111.2	320	58	1874.6	35.9	67232.5	1.0	0.6
Parser	Class	47.0	42.0	4.3	4005.5	7.0	1.3	110.8	930	98	6151.7	66.8	410900.7	6.3	2.1
PsiElement	Class	146.0	44.4	6.8	13921.1	12.2	1.0	97.5	2796	218	21719.8	140.2	3044201.3	47.0	7.2
Program	Class	42.0	33.6	3.4	3876.7	6.3	1.2	112.7	1036	140	7385.9	58.6	432459.4	6.7	2.5
Trash	Class	29.0	50.9	5.0	4142.4	7.0	0.9	110.6	460	56	2671.4	64.7	172845.7	2.7	0.9

Figura 8-8. Fragmento de las métricas detalladas de los elementos de programación de PsiEngine.

8.2.2.2 Resumen VOG del proyecto PsiEngine

En la Tabla 8-9 se muestra el resumen de características validadas en el proyecto PsiEngine, las cuales revalidan parcialmente el objetivo de investigación O2.

Tabla 8-9. Resumen VOG: Resumen de características validadas en el proyecto PsiEngine para PsiModel.

Nº	Características	Proyecto PsiEngine
5	Creación de elementos de programación usados en PsiModel	EC
6	Construcción de componentes usados en PsiModel	C

8.2.3 KERNEL: PROYECTO PSIMODEL

En esta sección se resumirá el análisis de las métricas para MPsi y MIPsi y se determina el aporte y la validación de las características para los objetivos PsiEngine y PsiModel. En el proyecto PsiModel en PsiEnvironment se especifican e implementan el lenguaje de Especificación MPsi y el lenguaje de Implementación MIPsi del PsiModel descritos en el Capítulo 4. En la Tabla 8-10 se resume este proyecto.

Tabla 8-10. Proyecto PsiModel en PsiEnvironment.

Proyecto	MPsi	MIPsi	#M	#LC	Dg.	Fichero	#LG	#LG/#LC
PsiModel (MPsi, MIPsi) *URL*: http://hilas.ii.uam.es/project?m=PsiEngine	1	6	7	1.166	☑	MPsi.js	2.339	2,0

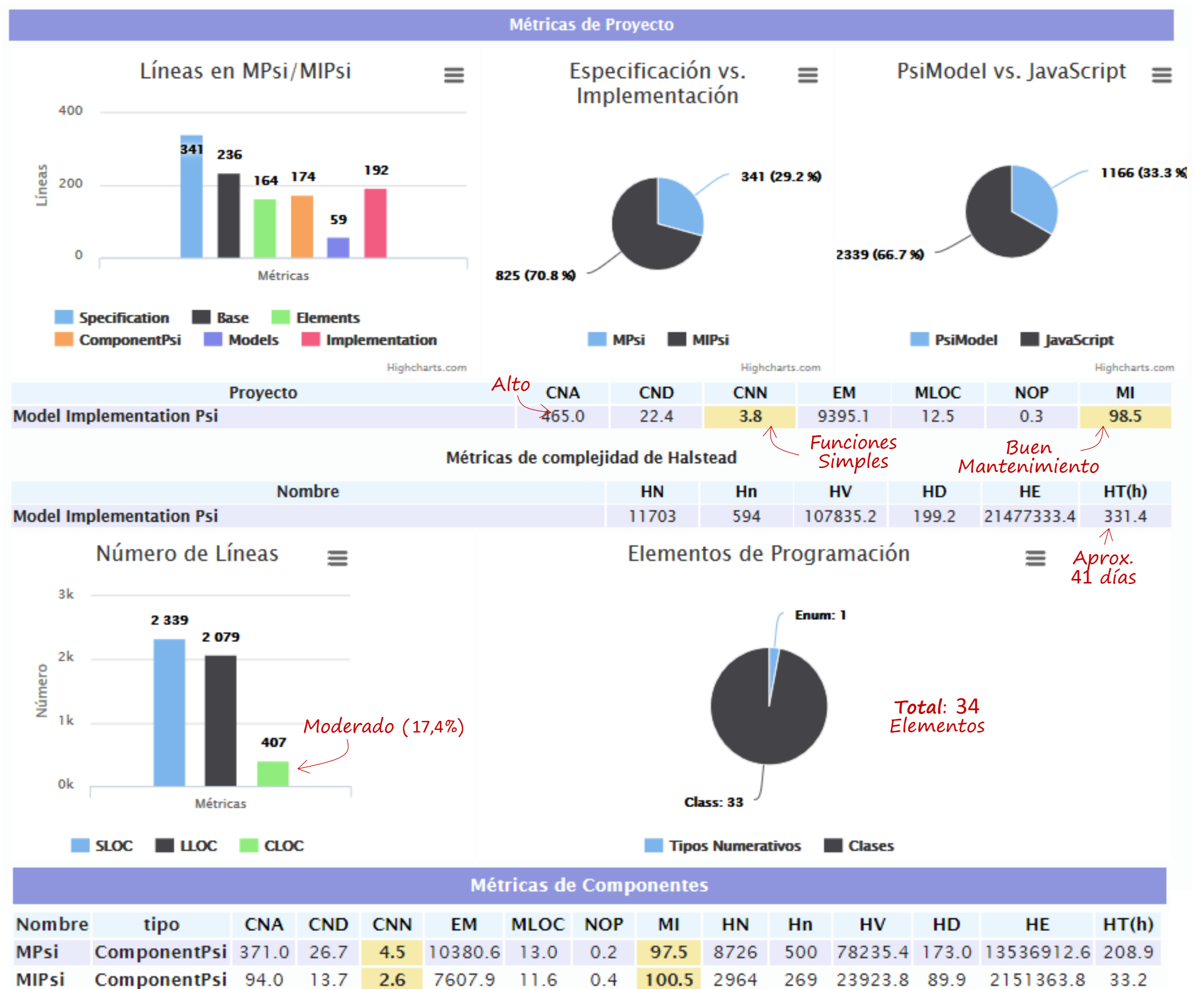
Nota: **MPsi**/**MIPsi**: Programas de PsiModel; **Dg.**: Diagrama de Clases Psi; **#M**: Total MPsi+MIPsi; **#LC**: Total de líneas de Psi; **Fichero**: Fichero generado; **#LG**: Líneas generadas JavaScript. Código fuente también disponible en DC adjunto.

8.2.3.1 Análisis de métricas

En la Figura 8-9 se muestran algunas de las métricas del proyecto de componentes PsiModel generadas por PsiEnvironment.

En estas métricas, la complejidad ciclomática promedio CNN de las funciones/métodos del proyecto muestra que se trata de “funcionalidades simples”, tiene buena capacidad de mantenimiento MI y las líneas de comentario CLOC son moderadas (17,4%). El tiempo estimado para implementar es de aproximadamente 41,4 días, donde 97% de los elementos de programación son clases. El total de elementos de programación es 22, del cual, el 55% (12) son clases. Estos resultados se obtienen analizando el proyecto en conjunto.

En la Figura 8-9, al analizar el compendio de métricas de componente Psi, el mayor esfuerzo promedio EM es para MPsi (10.381) siendo superior al MIPsi (7.608), pero es de notar que es aproximadamente 2 veces el vocabulario Hn y 3 veces la duración del programa HN de MPsi con respecto a MIPsi. Mientras que el número de parámetros promedio NOP y su mantenibilidad es buena para todos los elementos de programación.



Proyecto	CNA	CND	CNN	EM	MLOC	NOP	MI
Model Implementation Psi	465.0	22.4	3.8	9395.1	12.5	0.3	98.5

Nombre	HN	Hn	HV	HD	HE	HT(h)
Model Implementation Psi	11703	594	107835.2	199.2	21477333.4	331.4

Nombre	tipo	CNA	CND	CNN	EM	MLOC	NOP	MI	HN	Hn	HV	HD	HE	HT(h)
MPsi	ComponentPsi	371.0	26.7	4.5	10380.6	13.0	0.2	97.5	8726	500	78235.4	173.0	13536912.6	208.9
MIPsi	ComponentPsi	94.0	13.7	2.6	7607.9	11.6	0.4	100.5	2964	269	23923.8	89.9	2151363.8	33.2

Figura 8-9. Resumen de métricas del proyecto de componentes PsiModel.

Adicionalmente, las métricas por elemento de programación obtenidas en el proyecto PsiModel muestran que el número de errores estimados de todos los elementos de programación están por debajo del límite 2. Los mayores esfuerzos EM son los analizadores de lenguajes (clases *ModelPsiParser* y

ModelImplementationPsiParser), sus índices de mantenibilidad MI son moderados y su complejidad CNN es moderada, pero es de notar que todo su código es generado. Las clases con funcionalidad normal (CNN entre 5-10) son *Component* y *Base*, y el resto de clases tienen funcionalidad simple.

8.2.3.2 Resumen VOG del proyecto PsiModel

En la Tabla 8-11 se muestra el resumen de características validadas en el proyecto PsiModel que revalidan parcialmente los objetivos de investigación O1 y O2. La implementación básica de PsiModel se realizó *ad-hoc*, pero su refinamiento se hizo con PsiEnvironment.

Tabla 8-11. VOG: Resumen de características validadas en el proyecto PsiModel para PsiEngine y PsiModel.

Nº	Características	Proyecto PsiModel
1	Ejecución de programas Psi	M
2	Asociación de información heterogénea	X
3	Compartir objetos e información	M
4	Generación dinámica de una aplicación web por un programa Psi	☐
5	Creación de elementos de programación usados en PsiModel	EC
6	Construcción de componentes usados en PsiModel	P

8.2.4 Componentes de PsiDiagram: Proyectos GLPsi, DPsi, TPsi y WPPsi

En esta sección se resumirá el análisis de las métricas para GLPsi, DPsi, TPsi, WPPsi y DFPsi, y se determina lo que cada una de ellas aporta a la validación de los objetivos PsiEngine y PsiModel.

En la Tabla 8-12 se muestran los cuatro proyectos que conforman PsiDiagram (GLPsi, DPsi, TPsi y WPPsi) y el lenguaje adicional DFPsi que se creó para los proyectos SID y BPME de SCOOP. En estos proyectos se crearon 32 ficheros PsiModel, de los cuales el 15,6% (5) son de especificación Psi y el 84,4% (27) son de implementación Psi.

Se escribieron un total de 8.955 líneas de código PsiModel, con las que se generaron un total de 12.149 líneas de código JavaScript (es decir una concisión de 1,6). Los componentes de PsiDiagram tienen diagramas de clases y diagramas PsiLSD. Además, se crearon un total de 13 componentes de los cuales 5 son componentes Psi correspondientes a los lenguajes GLPsi, DPsi, TPsi, WPPsi y DFPsi.

Tabla 8-12. Proyectos de componentes de PsiDiagram en PsiEnvironment.

Caso de Estudio	MPsi	MIPsi	#M	#LC	Dg.	Fichero	#LG	#LG/#LC
GLPsi (Geometry, ConnectionEdge, Graph, Lines, GLPsi*) *URL*: http://hilas.ii.uam.es/project?m=GLPsi	1	7	8	2.904	☑	GLPsi.js	4.226	1,5
DPsi (Graph, Lines, DPsi*) *URL*: http://hilas.ii.uam.es/project?m=DPsi	1	5	6	1.924	☑	DPsi.js	3.014	1,6
TPsi (Help, Generator, TPsi*) *URL*: http://hilas.ii.uam.es/project?m=TPsi	1	9	10	2.762	☑	TPsi.js	4.201	1,5
WPPsi (WPPsi*) *URL*: http://hilas.ii.uam.es/project?m=WPPsi	1	4	5	813	☑	WPPsi.js	1.028	1,3
DFPsi (DFPsi*) *URL*: http://hilas.ii.uam.es/project?m=DFPsi	1	2	3	552	☐	DFPsi.js	1.149	2,1
Total	**5**	**27**	**32**	**8.955**			**13.149**	**1,6**

Nota: **MPsi**/**MIPsi**: Programas de PsiModel; **Dg.**: Diagrama de Clases Psi; **#M**: Total MPsi+MIPsi; **#LC**: Total de líneas de Psi; **Fichero**: Fichero generado; **#LG**: Líneas generadas JavaScript. (*): Componentes Psi. Código fuente también disponible en DC adjunto.

8.2.4.1 Análisis de métricas

En la Figura 8-10 se muestran las métricas para los proyectos de componentes de PsiDiagram generadas de forma automática por PsiEnvironment.

La complejidad ciclomática promedio CNN de las funciones/métodos para los proyectos GLPsi, TPsi, WPPsi y DFPsi indica que son son "funcionalidades simples", mientras que el proyecto DPsi son "funcionalidades normales". Todos los proyectos tienen buena capacidad de mantenimiento MI y el esfuerzo medio EM para implementar estos componentes es parecido con un promedio de 8.878.

El total SLOC es de 13.469 líneas JavaScript en todos los casos de estudio, de las cuales 17,4% en promedio son líneas comentarios y están en el rango moderado.

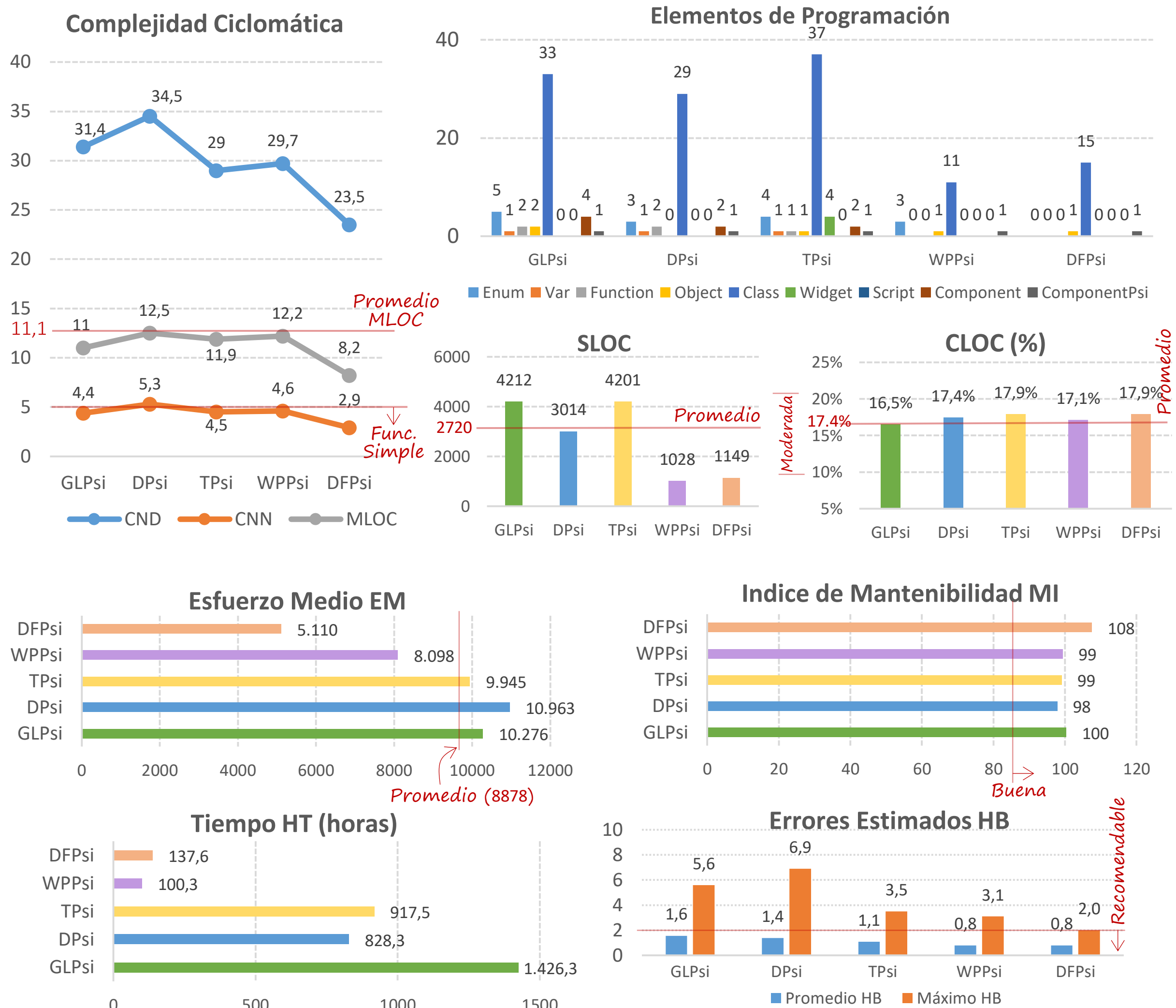


Figura 8-10. Resumen de métricas para los proyectos de PsiDiagram.

En el proyecto GLPsi, está conformado por 5 componentes (Geometry, ConnectionEdge, Graph, Lines y GLPsi) de los cuales uno es componente Psi (lenguaje GLPsi). Todos los componentes tienen funcionalidad simple, exceptuando Geometry que tiene funcionalidad normal (CNN=6,2) y sus MI son buenos. El total de elementos de programación es 159, de los cuales el 78,6% (125) son clases, el 7,5% (12) son tipos enumerativos, 3,1% (5) son objetos y el 10,8% (14) son de otro tipo. De estas clases, *GraphLibraryParser* tiene un CNN de 16 (complejidad intermedia, riesgo moderado) y MI de 76,4 (mantenibilidad moderada) que no están dentro de los parámetros deseados, aunque cabe notar que es una clase que se genera casi en su totalidad.

En el proyecto DPsi, está conformado por 3 componentes (Graph, Lines y GLPsi) de los cuales uno es componente Psi (lenguaje DPsi). El componente Graph tiene funcionalidad simple, mientras que Lines y DPsi tienen funcionalidad normal. Todos los componentes tienen buen índice de mantenimiento MI. El total de elementos de programación es 53, de los cuales el 79,3% (42) son clases. De estas clases, *DiagramParser* tiene un CNN de 26,7 (compleja, riesgo alto) y MI de 62,4 (difícil de mantener), pero nuevamente, está clase se genera casi en su totalidad.

En el proyecto TPsi, está conformado por 3 componentes (Help, Generator y GLPsi) de los cuales uno es componente Psi (lenguaje TPsi). El componente Generator tiene funcionalidad normal, mientras que Lines y DPsi tienen funcionalidad simple. Todos los componentes tienen buen índice de mantenimiento MI. El total de elementos de programación es 52, de los cuales el 80,8% (42) son clases. De estas clases, nuevamente el analizador *ToolPsiParser* tiene un CNN de 22,8 (compleja, riesgo alto) y MI de 67,8 (mantenibilidad moderada).

Por último, el proyecto TPsi está conformado por el componente Psi WPPsi (lenguaje WPPsi). El componente tiene funcionalidad normal y buen índice de mantenimiento MI. El total de elementos de programación es 16, de los cuales el 68,8% (11 clases) son clases. Cabe notar que la clase *WebPageParser* tiene funcionalidad normal (CNN=9,7) y buena mantenibilidad (MI=87,4).

El error promedio HB de los cinco proyectos este en niveles recomendables. Pero es de resaltar que en el proyecto DFPsi todos los elementos están niveles recomendables. Entre las clases a revisar por su promedio HB alto son: GLPsi (*CheckPoint* 5,6; *Line* 5,1; y *Segment* 5,1), DPsi (*ManagerDiagram* 6,9; *GraphBase* 5,1; y *DCommon* 5,6), TPsi (*ConteinerTool* 3,5) y WPPsi (*Content* 3,1).

8.2.4.2 Resume VOG para los componentes PsiDiagram

En la Tabla 8-13 se muestra el resumen de características validadas por los proyectos GLPsi, DPsi, TPsi, WPPsi y DFPsi concernientes a los objetivos relacionados con PsiEngine y PsiModel. Cabe notar que con estos proyectos se validó nuevamente casi la totalidad de las características para los objetivos de investigación O1 y O2.

Tabla 8-13. VOG: Resumen de características validadas en los proyectos GLPsi, DPsi, TPsi, WPPsi y DFPsi para PsiEngine y PsiModel.

Nº	Características	Proyectos				
		GLPsi	DPsi	TPsi	WPPsi	PFPsi
1	Ejecución de programas Psi	P	M	P	M	S
2	Asociación de información heterogénea	X	XJ	XJ	XJ	X
3	Compartir objetos e información	M	M	M	M	M
5	Creación de elementos de programación usados en PsiModel	VFEOC	VFEC	VFEOCW	EOC	OC
6	Construcción de componentes usados en PsiModel	CP	CP	CP	CP	P

8.2.5 Componentes Móviles: Proyectos SMAPPS

El **Proyecto SMAPPS** es un proyecto que contiene un conjunto de programas Psi para la creación y edición de aplicaciones móviles basadas en BPME de SCOOP. En la sección 7.5, se explicó el diagrama de bloques para los proyectos BPMEPsi, SMAPPS y AngularSMAPPS, donde los proyectos BPMEPsi y SMAPPS crean una aplicación móvil basada en servicios SMAPPS y el framework jQuery Mobile. Por su parte AngularSMAPPS crea una aplicación móvil basada en servicios SMAPPS, AngularJS y Angular Material.

En esta sección se resumirá el análisis de las métricas para AngularSMAPPS, versión actual de aplicaciones SMAPPS de B2T SCOOP. Después, se hará una comparativa entre los proyectos de AngularSMAPPS, BPMEPsi y SMAPPS. Por último, se determina el aporte a la validación de los objetivos PsiEngine y PsiModel.

8.2.5.1 Análisis de métricas para AngularSMAPPS

En la Tabla 8-14 se muestran los cuatro proyectos que conforman AngularSMAPPS: AngularSMAPPSCore, AngularSMAPPSRender, AngularSMAPPSRun y AngularSMAPPSUtils.

En los proyectos se crearon 18 ficheros PsiModel, de los cuales el 22,2% (4 ficheros) son de especificación Psi y el 78,8% (14 ficheros) son de implementación Psi. Se escribieron un total de 3.796 líneas de código PsiModel, con las que se generaron un total de 5.266 líneas de código JavaScript (es decir a una concisión de 1,4). Cabe destacar la concisión 1,9 sobre el componente BPMEPsi, dado que se implememnta de forma abstracta el BPMN. Solo se crearon diagramas de clases y diagramas PsiLSD para AngularSMAPPSCore y diagrama de clases para AngularSMAPPSRender.

Tabla 8-14. Proyectos AngularSMAPPS.

Casos de Estudios	MPsi	MIPsi	#M	#LC	Dg.	Fichero	#LG	#LG/#LC
AngularSMAPPSCore (BPMEPsi*, Implement). *URL*: http://hilas.ii.uam.es/project?m=AngularSMAPPSCore	1	2	3	818	☑	SMAPPSCore.js	1.592	1,9
AngularSMAPPSRender (View, Controls).*URL*: http://hilas.ii.uam.es/project?m=AngularSMAPPSRender	1	7	8	1.496	☑	SMAPPSRender.js	1.967	1,3
AngularSMAPPSRun. *URL*: http://hilas.ii.uam.es/project?m=AngularSMAPPSRun	1	2	3	444		SMAPPSRun.js	485	1,1
AngularSMAPPSUtils. *URL*: http://hilas.ii.uam.es/project?m=AngularSMAPPSUtils	1	3	4	1.038		SMAPPSUtils.js	1.222	1,2
Total	**4**	**14**	**18**	**3.796**			**5.266**	**1,4**

Nota: **MPsi/MIPsi**: Programas de PsiModel; **Dg.**: Diagrama de Clases Psi; **#M**: Total MPsi+MIPsi; **#LC**: Total de líneas de Psi; **Fichero**: Fichero generado; **#LG**: Líneas generadas JavaScript. (*): Componentes Psi. Código fuente también disponible en DC adjunto.

En la Figura 8-11 se muestra el resumen de métricas para los proyectos AngularSMAPPS. En estas métricas, la complejidad ciclomática promedio CNN de las funciones/métodos para los proyectos AngularSMAPPS muestra que se tratan de "funcionalidades simples". Todos los proyectos tiene buena capacidad de mantenimiento MI y las líneas de comentario CLOC son moderadas (21,2% en promedio), es el mayor promedio de todos casos de estudio vistos.

El esfuerzo medio EM para implementar estos proyectos es 4.165, cabe notar que la suma de EM AngularSMAPPSRender y AngularSMAPPSCore, es aproximadamente el mismo que AngularSMAPPSUtils.

En el proyecto AngularSMAPPSCore, está integrado por el componente Implement y el componente Psi BPMEPsi (lenguaje BPMEPsi). Todos los componentes tienen funcionalidad simple y sus MI son buenos. El total de elementos de programación es 38, de los cuales el 94,7% (36) son clases. De estas clases, *BPMExecutionParser* tiene un CNN de 8 (función/programa normal, riesgo normal) y su MI de 92,1 (buen mantenimiento) que no están dentro de los parámetros deseados, pero cabe notar que es una clases que se genera casi en su totalidad. En este proyecto todos sus errores estimados están en el rango recomendable.

En el proyecto AngularSMAPPSRender, está contiene por los componentes View y Controls. Ambos componentes tienen funcionalidad simple y buen índice de mantenimiento MI. El total de elementos de programación es 33, de los cuales el 67,7% (21) son clases, el 22,6% (7) son objetos, y el 15,2% (5) de otro tipo. De estos elementos, el objeto *Factory* es el único que tiene funcionalidad normal, el resto de elementos, tienen funcionalidad simple. Además, *Factory* su MI es 89,8 dentro de lo recomendable.

En el proyecto AngularSMAPPSRun, es un proyecto de configuración y está conformado por 3 objetos y una función. Tienen funcionalidad simple y buen índice de mantenimiento MI.

En el proyecto AngularSMAPPSUtils, está conformado por 11 elementos de programación, de los cuales el 36,3% (4) son clases, y el 45,5% (5) son objetos y el 18,2% (2) de otro tipo. De estos elementos, el objeto *Board* (la *pizarra* del servicio) es el único que tiene funcionalidad normal, el resto son funcionalidad simple. Además, el objeto *Board* tiene MI de 87,6 dentro de lo recomendable.

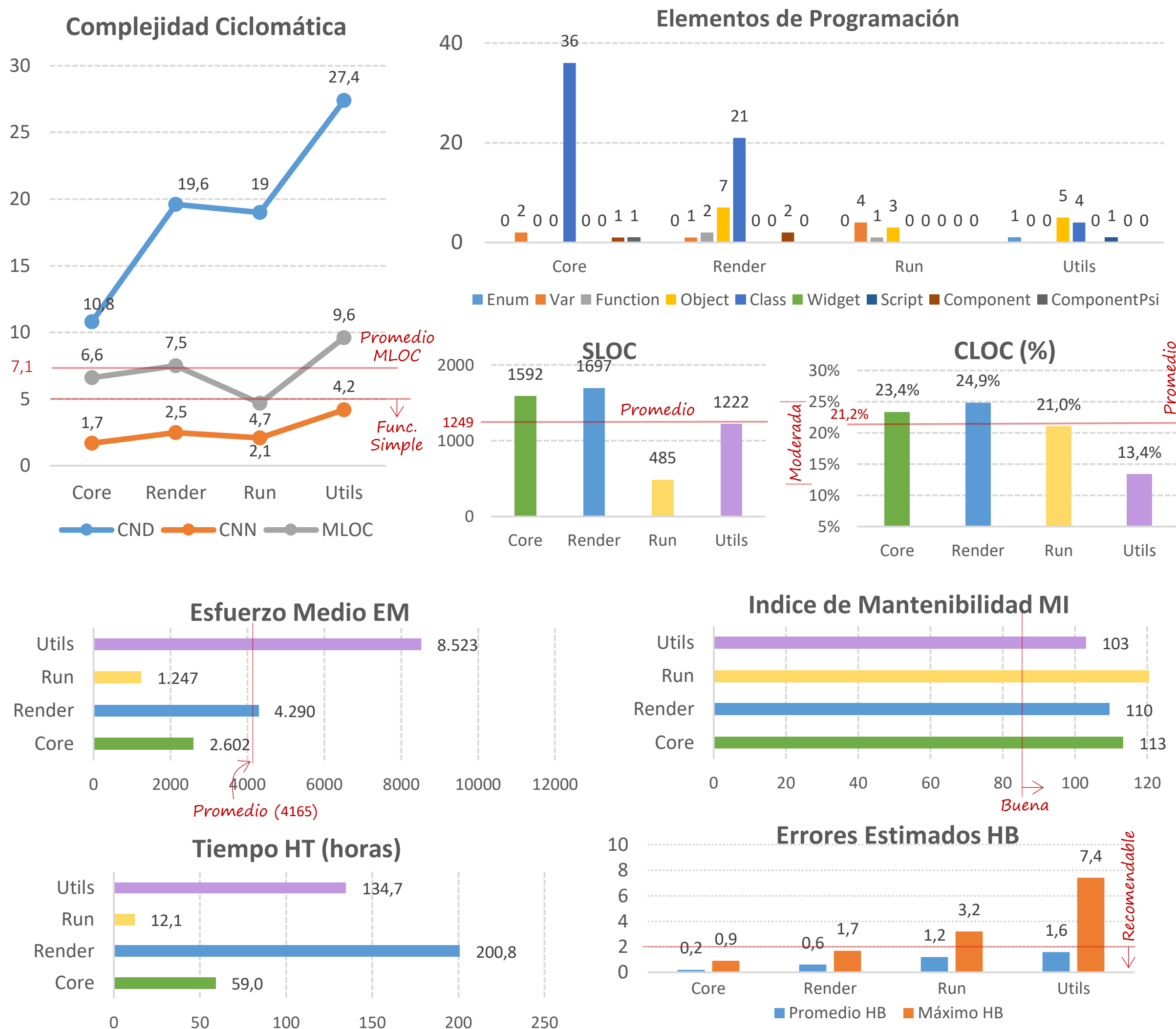


Figura 8-11. Resumen de métricas para los proyectos AngularSMAPPS.

El error promedio HB de los cuatro proyectos este en niveles recomendables. Entra las clases y objetos a revisar por su promedio HB alto son: AngularSMAPPSCore (*ninguno*), AngularSMAPPSRender (*ninguno*), AngularSMAPPSRun (*Execute* 3,2) y AngularSMAPPSUtils (*Board* 7,4).

8.2.5.2 Comparación de métricas de los proyectos BPMEPsi, SMAPPS y AngularSMAPPS

En la sección anterior se mostraron las métricas del proyecto AngularSMAPPS, en la Figura 8-12 se comparan las métricas de los proyectos BPMEPsi, SMAPPS y AngularSMAPPS.

La complejidad ciclomática CNN para los tres proyectos es muy parecida, de funcionalidad simple. Pero la cantidad de líneas en AngularSMAPPS es aproximadamente un 20% menos que los proyectos BPMEPsi y SMAPPS. Además, el esfuerzo medio EM de AngularSMAPPS es aproximadamente 36% menor que los otros proyectos. El índice de mantenibilidad es ligeramente mejor (112) en AngularSMAPPS. El tiempo HT decrece en el proyecto AngularSMAPPS en un 63% con respecto a la primera versión con el proyecto BPMEPsi.

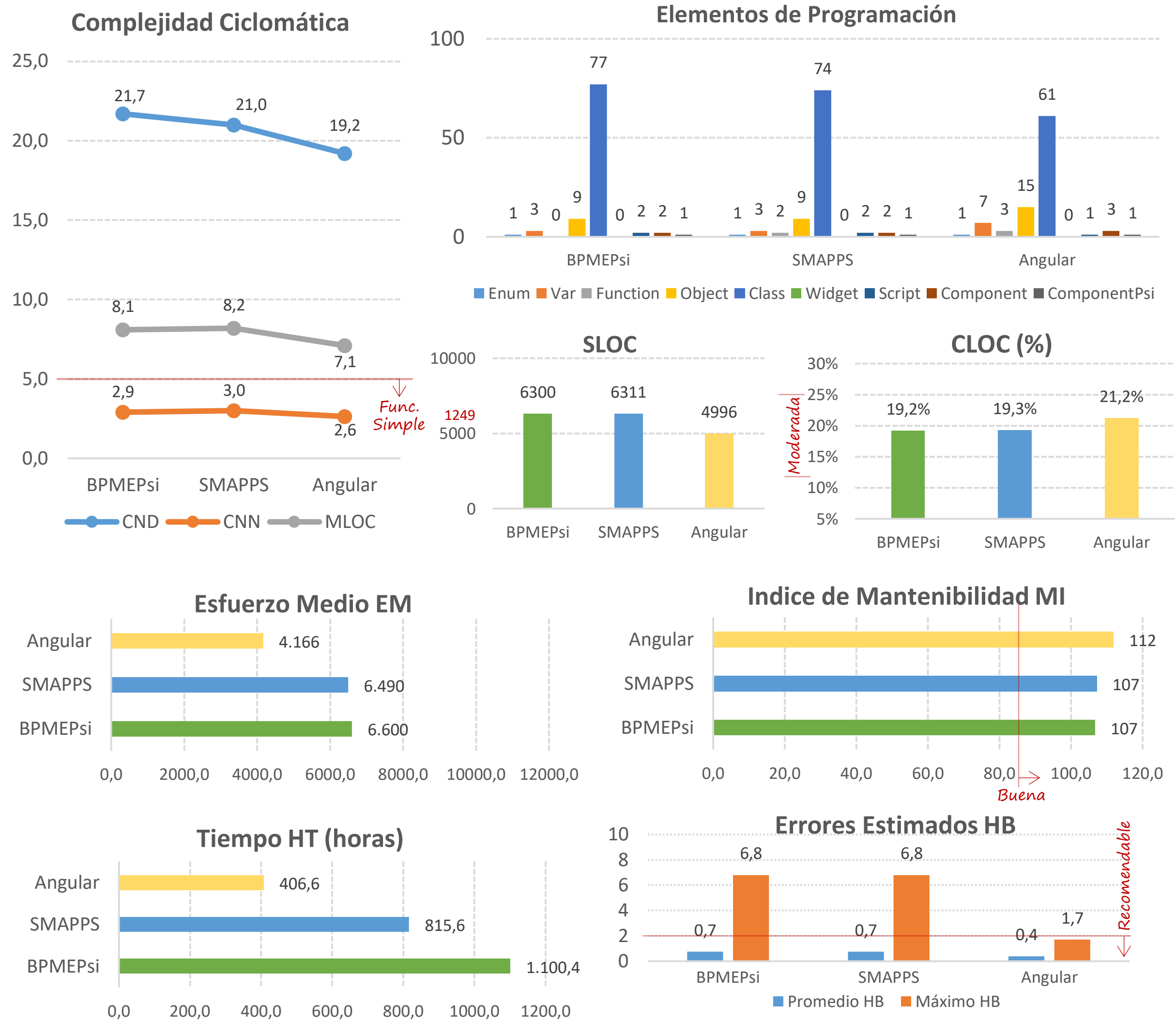


Figura 8-12 Métricas comparativas de los proyectos BPMEPsi, SMAPPS y AngularSMAPPS.

También es meritorio decir que la estimación de errores promedios HB en AngularSMAPPS logra niveles recomendables para la creación de estos componentes.

En conclusión, se ha logrado obtener una mejor *aplicación móvil SMAPPS* con el proyecto AngularSMAPPS, y se debe en gran medida a la experiencia preliminar y a un mejor diseño e implementación.

8.2.5.3 Resumen VOG de los componentes móviles

En la Tabla 8-15 se muestra el resumen de características validadas por los proyectos BPMEPsi, SMAPPS, AngularSMAPPS concernientes a los objetivos relacionados con PsiEngine y PsiModel.

Cabe notar que con estos proyectos se revalida nuevamente casi la totalidad de las características para los objetivos de investigación O1 y O2. Además, los proyectos revalidan de forma muy importante la generación dinámica de aplicaciones web usando programas Psi. También se presenta la perfecta combinando del uso de lenguajes DSVL con estándares como lo es BPMN 2.0, para la creación de aplicaciones móviles híbridas.

Tabla 8-15. VOG: Resumen de características validadas en los proyectos de BPMEPsi, SMAPPS, AngularSMAPPS para PsiEngine y PsiModel.

Nº	Características	Proyectos		
		BPMEPsi	SMAPPS	Angular SMAPPS
1	Ejecución de programas Psi	P	P	P
2	Asociación de información heterogénea	XJ	XJ	XJ
3	Compartir objetos e información	S	S	S
4	Generación dinámica de una aplicación web por un programa Psi	☑	☑	☑
5	Creación de elementos de programación usados en PsiModel	VFEOC	VFEOC	VFEOC
6	Construcción de componentes usados en PsiModel	CP	CP	CP

8.2.6 Notas finales de validación

La validación de esta investigación ha seguido la metodología cualitativa de casos de estudio (tipo multicaso) sugerida por (Yin, 2014) y adaptada para la Ingeniería de Software en (Baxter & Jack, 2008). Se concreta en conjunto de características a validar. Aunque fue extremadamente costoso en tiempo y ejecución, se logró tener un conjunto de 20 casos de estudio (proyectos PsiModel de la Tabla 8-3) que validan múltiples veces los objetivos PsiEngine y PsiModel enumerados en la Tabla 8-1 de *VOG*. En estos casos de estudio se crearon 114 programas Psi (20 programas en lenguaje de Especificación MPsi, y 94 programas en lenguaje de Implementación MIPsi) y 114 ficheros de recursos, para crear el Intérprete PsiXML, 16 componentes Psi, 18 componentes JavaScript y 8 widgets web, para las diferentes aplicaciones web y aplicaciones móviles.

En la Tabla 8-16 se resume aproximadamente cuántas veces se han validado cada uno de las características de PsiEngine y PsiModel. En esta tabla se adicionaron las validaciones del proyecto UPsi que no se registró ningún tipo de análisis de métricas.

También cabe recordar que del total de 20 proyectos, el 75% (15) están en producción, y de estos, en SCOOP están funcionando 7 proyectos, mientras que en aplicaciones móviles funcionan 8 proyectos.

Los proyectos en producción ratifican la validez de las características validadas. Cada proyecto supera las pruebas unitarias realizadas por el desarrollador. Seguido, se pasa a un entorno de pruebas de SCOOP, que es exactamente el mismo de producción para validar los proyectos. Aprobados los criterios de calidad en B2T Concept pasa a producción.

En el caso de los Componentes Móviles CM, las Aplicaciones Móviles SMAPPS se proban en dispositivos con sistema operativo Android e iOS, antes de ser puestas en producción.

Tabla 8-16. VOG: Número de validaciones de las características validadas para PsiEngine y PsiModel.

		Características	Número de Validaciones
PsiEngine	1	Ejecución de programas Psi: (S) simple-simple (un programa de un lenguaje Psi); (P) múltiple-simple (múltiples programas de un lenguaje Psi); (M) múltiple-múltiple (múltiples programas escritos en múltiples lenguajes Psi)	S =1; P=7; M=6
	2	Asociación de información heterogénea: (X) XML; (J) JSON	X=13; J=8
	3	Compartir objetos e información entre: (S) diferentes programas escritos en el mismo lenguaje Psi; (M) diferentes programas escritos en diferentes lenguajes Psi	S=4; M=9
	4	Generación dinámica de una aplicación web por un programa Psi	3
PsiModel	5	Creación de elementos de programación usados en PsiModel: (V) variable; (F) función; (E) tipo enumerado; (O) Objeto; (C) Clase; (W) Web widget;	V=13;F=13;E=21;O=46; C=419; W=10
	6	Construcción de componentes usados en PsiModel: (C) crear componente JavaScript; (P) crear componentes Psi (es decir, crear lenguajes XML-DSLs)	C=19; P=16;

8.3 VALIDACIÓN DE OBJETIVOS DE PSIDIAGRAM

Esta sección está dedicada a validar las características de PsiDiagram en relación con el objetivo general de esta tesis, descrito en la metodología de validación de la sección 8.1 (Tabla 8-1). Para ello, en la Tabla 8-18 se muestra un resumen de la lista de proyectos PsiDiagram desarrollados en PsiEnvironment, que nos sirven como casos de estudio. Están clasificados de la siguiente forma:

i. **Diagrama de clases y diagramas PsiLSD**: proyectos con el objetivo de crear y editar los diagramas de clases y diagramas de PsiLSD, respectivamente.

ii. **Generación automática de diagramas ADG**: el Proyecto ADG de SCOOP tiene como objetivo la generación de diagramas como modelo de estados, diagramas de GANTT, diagramas de PERT, modelo de datos, árboles jerárquicos WBT, diagramas de SCOOP (llamado de funciones, mapa de objetos, modelo de conceptos-relaciones).

iii. **Diagramas SID**: el Proyecto SID (SCOOP Interface Diagram) de SCOOP tiene como objetivo la creación y edición de los modelos conceptos-relaciones, modelo de estados, modelos de procesos y BPM para SCOOP.

iv. **Diagramas BPME**: el Proyecto BPME de SCOOP tiene como objetivo la creación y edición de BPMN 2.0 con PsiDiagram. Sus aplicaciones prácticas son la implementación de DSVL para la creación de aplicaciones móviles SMAPPS, la generación automática de BPM de SCOOP y el visor de proyectos BPME.

Tabla 8-17. Lista de proyectos PsiDiagram para la creación de diagramas, modelos gráficos y DVSLs.

Clasificación	Proyecto	GD	RD	DF	DSVL	DES	PRD	Prg.Psi	#Psi	Otros	#Otros	Total	#Total
Diagrama de clases y PsiLSD	PsiBase					☑		5	1.011	3	269	8	1.280
	PsiClassDiagram					☑		13	822	0	0	13	822
	PsiClassBasic					☑		13	809	0	0	13	809
	PsiLSD					☑		17	704	0	0	17	704
Generación automática de diagramas ADG	ADGBase						☑	5	1.634	1	59	6	1.693
	ADG-ES	☑	☑				☑	5	437	0	0	5	437
	ADG-FU	☑					☑	5	350	0	0	5	350
	ADG-GA	☑					☑	5	696	1	27	6	723
	ADG-MD	☑	☑				☑	5	481	0	0	5	481
	ADG-MP	☑					☑	5	350	0	0	5	350
	ADG-PE	☑					☑	5	702	0	0	5	702
	ADG-RE	☑					☑	5	488	0	0	5	488
	ADG-RX	☑					☑	5	488	0	0	5	488
	ADG-WB	☑					☑	5	312	0	0	5	312
Diagramas SID	SIDBase						☑	11	2.710	2	233	11	2.943
	SID-CR	☑		☑			☑	5	309	1	27	5	336
	SID-SR	☑		☑			☑	5	350	1	26	5	376
	SID-PM	☑		☑			☑	6	598	1	39	6	637
	SID-SCR	☑		☑			☑	5	344	1	31	5	375
	SID-SSR	☑		☑			☑	5	352	1	26	5	378
	SID-SBPM	☑		☑			☑	5	1.155	1	77	5	1.232
Diagramas BPME	BPMEBase						☑	12	3.345	1	405	13	3.144
	BPMVisor			☑			☑	3	223	5	404	3	627
	BPMSMAPPS				☑		☑	2	128	1	114	2	302
	BPMSCOOP	☑		☑			☑	4	1.442	1	90	4	1.532
Total		**16**	**2***	**8**	**1**	**4**	**21**	**161**	**20.300**	**21**	**1.827**	**182**	**22.127**

Nota: **Proyecto**: Proyecto de PsiDiagram; **GD**: Generador de Diagramas; **RD***: Prototipo Recomendador de Diagramas; **DF**; Diagrama Funcional; **DES**: Casos de estusio en desarrollo; **PRD**: Casos de estudio en producción; **Prg.Psi**: Total de programas Psi (GLPs+DPsi+TPsi+WPPsi+DFPsi); **#LCPsi**: Número de líneas Psi; **Otros**: Otros ficheros.

Como se puede ver en la Tabla 8-3, actualmente en PsiEnvironment se tiene un total de 25 proyectos de PsiDiagram, de los cuales el 19% (4) son diagrama se soporte al desarrollo y el 81% (21) son diagramas que están en producción.

De los proyectos en producción en SCOOP, el 76% (16) diagramas implementa un Generador de Diagramas, para crear diagramas o redistribuir diagramas de forma automática. Por otro lado, el 35% (8) tiene como objetivo la sincronización entre la edición de diagramas y la información preexistente en la plataforma SCOOP.

Para PsiDiagram, se han escrito un total de 161 ficheros en los diferentes lenguajes GLPs, DPsi, TPsi, WPPs y DFPsi, y se han escrito 20.300 líneas de código Psi. En la Figura 8-14 se muestra la distribución de número de líneas en las diferentes clasificaciones.

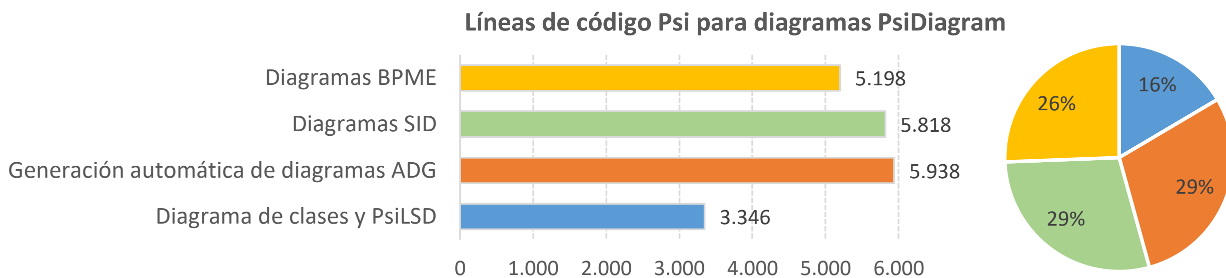


Figura 8-13. Líneas de código Psi escritas en GLPs, DPsi, TPsi, WPPs y DFPsi para PsiDiagram.

8.3.1 Diagramas de Clases y Diagramas PsiLSD

En la Tabla 8-18 se muestran los proyectos PsiBase, PsiClassDiagram, PsiClassBasic y PsiLSD para la creación y edición de diagramas de clases y diagramas de PsiLSD. Se distinguirá entre dos tipos de diagramas de clases: el *detallado* (incluye propiedades, métodos y visualización de código de las clases) y *básico* (solo nombre de clases). En estos proyectos se crearon 51 ficheros, de los cuales el 86,3% (44) son de diagramación Psi, es decir, 13 diagramas de clases, 13 diagramas de clases básicos y 17 diagramas PsiLSD. Para ver el detalle de los programas Psi y sus métricas, visite cada proyecto PsiDiagram.

Tabla 8-18. Proyectos diagramas de clases y diagramas PsiLSD de tipo PsiDiagram en PsiEnvironment.

Caso de Estudio	GLPsi	DPsi	TPsi	WPPsi	DFPsi	#Psi	Otros	Total
PsiBase (Creación y edición de diagramas de clases y diagramas PsiLSD) *URL*: http://hilas.ii.uam.es/project?d=PsiBase	1 #26	1 #350	2 #434	1 #26	0 #0	5 #1.011	3 #269	8 #1.280
PsiClassDiagram (Diagrama de clases) *URL*: http://hilas.ii.uam.es/project?d=PsiClassDiagram	0 #0	13 #822	0 #0	0 #0	0 #0	13 #822	0 #0	13 #822
PsiClassBasic (Diagrama de clases básico) *URL*: http://hilas.ii.uam.es/project?d=PsiClassBasic	0 #0	13 #809	0 #0	0 #0	0 #0	13 #809	0 #0	13 #809
PsiLSD (Diagrama PsiLSD) *URL*: http://hilas.ii.uam.es/project?d=PsiLSD	0 #0	17 #704	0 #0	0 #0	0 #0	17 #704	0 #0	17 #704
Total número de ficheros **Total número de líneas**	**1** **#26**	**44** **#2.536**	**2** **#434**	**1** **#26**	**0** **#0**	**48** **#3.346**	**3** **#269**	**51** **#3.615**

Nota: **GLPsi\DPsi\TPsi\WPPsi\DFPsi**: programas Psi / # líneas Psi; **#Psi**: total programas Psi / # total líneas código Psi; **Otros**: Otros ficheros / # líneas; **Total**: #Psi+Otros. **Código fuente también disponible en DC adjunto.**

8.3.1.1 Editor de diagramas de clases y PsiLSD

En la Figura 8-14 se muestra el editor de diagrama de clases y diagramas PsiLSD basado en el proyecto **PsiBase** de PsiDiagram, que denominaremos **PsiEditDiagram**.

Figura 8-14. Editor de diagrama de clases y diagramas PsiLSD basado en el proyecto PsiBase de PsiDiagram.

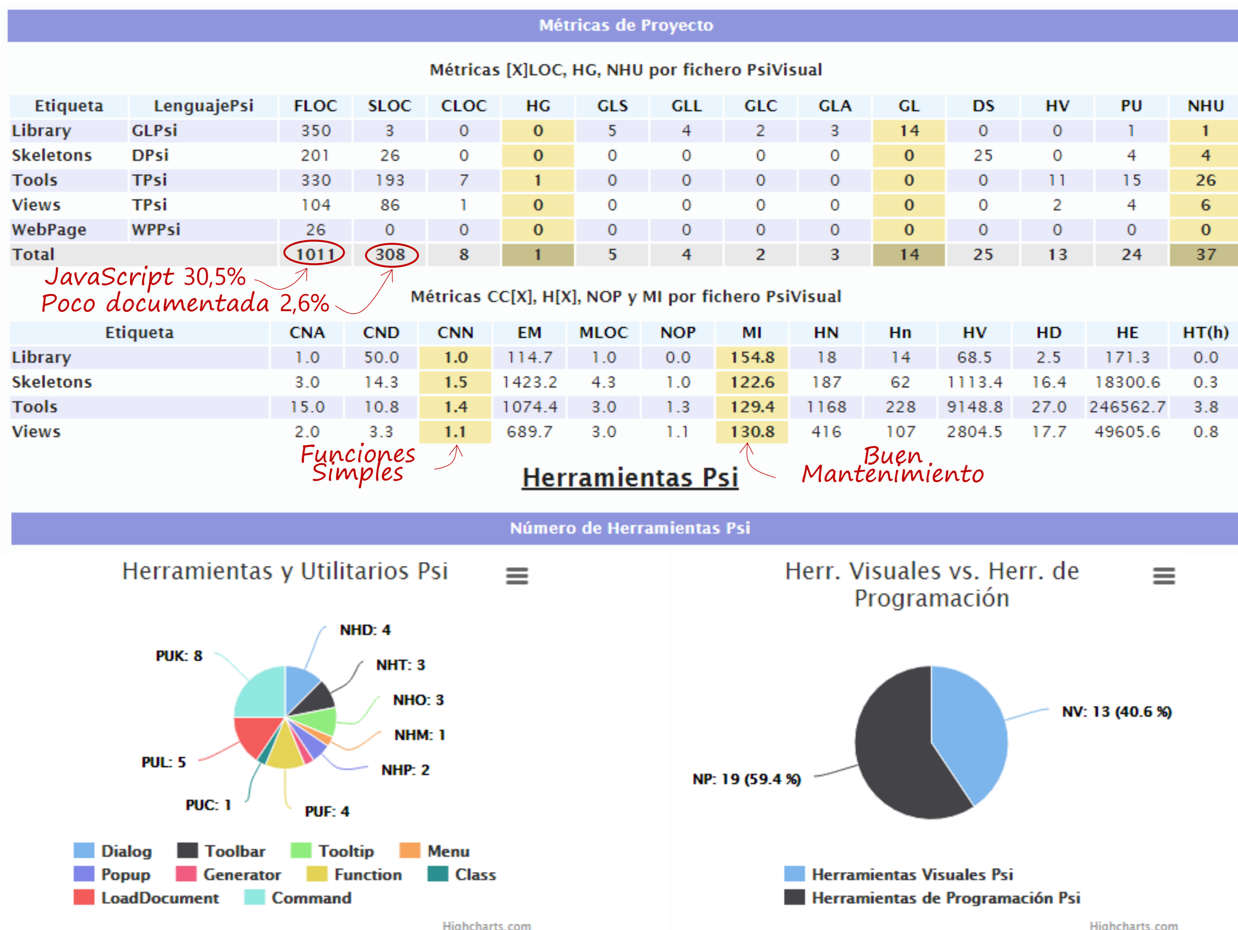
El PsiEditDiagram es una aplicación web de una página. Tiene dos tipos de visores: *Visor EPPsi* y *Visor de Código JS*. El *Visor EPPsi* muestra información de las variables, funciones, objetos y clases, por ejemplo, el *Visor de Clase* muestra las propiedades, eventos, métodos y métodos estáticos de la clase. Por último, el *Visor de Código JS* muestra el código de variables, propiedades y cuerpos de métodos, eventos y funciones, de los diferentes elementos de programación Psi.

Con el PsiEditDiagram se pueden ver algunas de las implementaciones en línea de los proyectos PsiModel de la Tabla 8-3, es decir, actualizado a la última versión (solo para los casos de estudio de PsiModel, para los proyectos en producción de SCOOP están en un ambiente controlado de desarrollo). Además, se usó para representar todos los diagramas de clases y diagramas de PsiLSD de esta tesis.

Por último, el PsiEditDiagram en modo visor se usa para la visualización de diagrama de clases y diagramas de PsiLSD en la ayuda en línea de los diferentes componentes en la página web PsiEngine (http://hilas.ii.uam.es). El código fuente también está disponible en el CD adjunto.

8.3.1.2 Análisis de métricas

En la Figura 8-16 se muestra el resumen de las métricas PsiDiagram para el proyecto PsiBase. Implementa 5 programas Psi (GLPsi=1, DPsi=1, TPsi=2 y WPPsi=1). Del total de líneas FLOC, el 30,5% son líneas de código JavaScript asociadas a la lógica de las diferentes herramientas y está poco documentado (CLOC 2,6%).



Métricas de Proyecto

Métricas [X]LOC, HG, NHU por fichero PsiVisual

Etiqueta	LenguajePsi	FLOC	SLOC	CLOC	HG	GLS	GLL	GLC	GLA	GL	DS	HV	PU	NHU
Library	GLPsi	350	3	0	0	5	4	2	3	14	0	0	1	1
Skeletons	DPsi	201	26	0	0	0	0	0	0	0	25	0	4	4
Tools	TPsi	330	193	7	1	0	0	0	0	0	0	11	15	26
Views	TPsi	104	86	1	0	0	0	0	0	0	0	2	4	6
WebPage	WPPsi	26	0	0	0	0	0	0	0	0	0	0	0	0
Total		1011	308	8	1	5	4	2	3	14	25	13	24	37

Métricas CC[X], H[X], NOP y MI por fichero PsiVisual

Etiqueta	CNA	CND	CNN	EM	MLOC	NOP	MI	HN	Hn	HV	HD	HE	HT(h)
Library	1.0	50.0	1.0	114.7	1.0	0.0	154.8	18	14	68.5	2.5	171.3	0.0
Skeletons	3.0	14.3	1.5	1423.2	4.3	1.0	122.6	187	62	1113.4	16.4	18300.6	0.3
Tools	15.0	10.8	1.4	1074.4	3.0	1.3	129.4	1168	228	9148.8	27.0	246562.7	3.8
Views	2.0	3.3	1.1	689.7	3.0	1.1	130.8	416	107	2804.5	17.7	49605.6	0.8

Figura 8-15. Resumen de métricas del proyecto PsiBase.

La complejidad ciclomática promedio CNN de las funciones/métodos del código JavaScript CLOC del proyecto PsiBase señalan que se trata de "funcionalidades simples" y tiene buena capacidad de mantenimiento MI, en promedio 134. Es de notar que el esfuerzo EM es el 75,6% para los programas *Skeletos* y *Tools*, pero el de mayor dificultad HD es *Tools*.

El proyecto PsiBase tiene un total de 37 herramientas Psi de las cuales el 64,8% (19 herramientas en programas TPsi, 4 en programas DPsi y una en programas GLPsi) son elementos de programación, y el resto 35,2% (13) son herramientas visuales. Dentro de estas herramientas, dispone de las cajas de diálogo para el editor de diagramas Psi, el visor EPPsi, así como de la ventana emergente del visor de código JavaScript del PsiEditDiagram. Adicionalmente, crea una clase *Editor* para implementar la funcionalidad del editor de diagramas Psi. Los diagramas circulares de la Figura 8-16 muestran el compendio de herramientas Psi, solamente de los programas TPsi.

Por último, en el programa GLPsi *Library* y el programa DPsi *Skeletons* se crean 14 elementos gráficos Psi y 25 skeletons con los cuales se da soporte a la creación de diagramas de clases, diagramas de clases básicos y diagramas PsiLSD. Es de notar que el código JavaScript (308 líneas) es bastante reducido para incorporar la funcionalidad que tiene PsiEditDiagram.

En la Figura 8-16 se muestra el número de elementos gráficos de la diagramación Psi de los proyectos PsiClassDiagram, PsiClassBasic y PsiLSD. Se han creado un total de 1.725 elementos gráficos, que aproximadamente están repartidos en forma equitativa en los 3 proyectos: PsiClassDiagram (577 elementos, 33,6%), PsiClassBasic (558 elementos, 32,5%) y PsiLSD (580 elementos, 33,8%). En referencia a elementos gráficos, el 80,6% (478 figuras y 904 líneas) son figuras y líneas Psi.

8.3.1.3 Resumen VOG para los diagramas de clases y los PsiLSD

En la Tabla 8-19 se muestra el resumen de características validadas por los proyectos PsiBase, PsiClassDiagram, PsiClassBasic y PsiLSD concernientes a los objetivos relacionados con PsiDiagram. Este proyecto nos ayuda a validar en gran medida la funcionalidad básica de la diagramación Psi, es decir, definir una librería gráfica Psi y usarla para la diagramación Psi de diversos modelos gráficos.

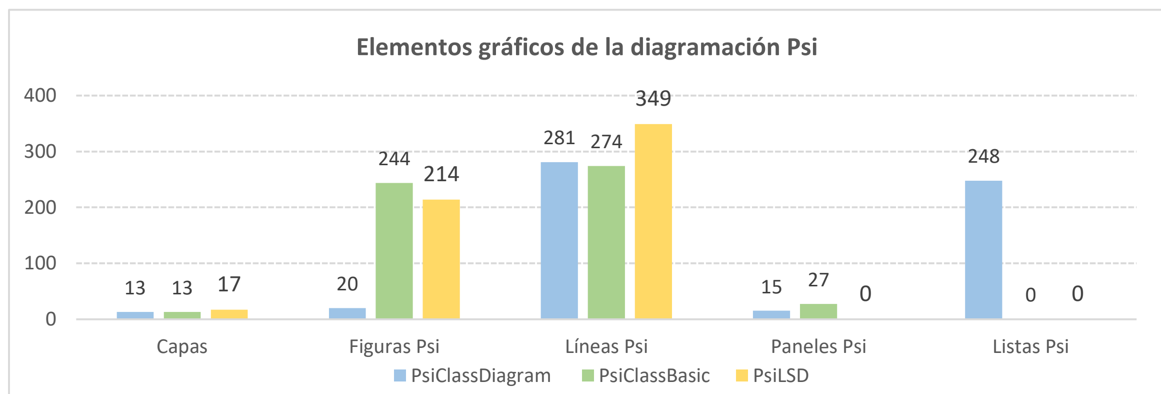


Figura 8-16. Elementos gráficos de la diagramación Psi de los proyectos PsiClassDiagram, PsiClassBasic y PsiLSD.

En particular, se han definido en una librería gráfica Psi 14 elementos gráficos de todos los elementos disponibles en PsiDiagram (Línea Psi S=5, Figura Psi L=4, Contenedor Psi C=2 y Lista Psi A=3), dentro del proyecto PsiBase. Con la aplicación web PsiEditDiagram se crearon hasta el momento 44 diagramas (entre diagramas de clase y diagramas PsiLSD) con los que se obtuvieron 1.715 elementos gráficos diagramados (nodos Psi S*=478, conectores L*=904, paneles C*=42; listas A*=248).

Tabla 8-19. VOG: Resumen de características/funcionalidades validadas con PsiEditDiagram.

Nº	Características	Proyectos	
		PsiBase	PsiClassDiagram, PsiClassBasic y PsiLSD
7	Creación de diagramas basados en GLPsi y DPsi	S=5; L=4; C=2; A=3; F=1; E=25	S*=478; L*=904; C*=42; A*=248
8	Creación de herramientas asociadas con TPsi	HV (D=4; B=3; F=1; T=3; P=2, M=1). PU (F=4; C=1; K=8; L=5)	-

Nota: (*) validación del uso de elementos gráficos Psi en la diagramación Psi.

8.3.2 Proyecto ADG: Generación Automática de Diagramas

En esta sección resumimos los tipos de diagrama **TypeADG** en el Proyecto ADG de SCOOP de generación automática de diagramas basados (ver sección 7.2). Se analizará las métricas PsiDiagram del proyecto ADGBase y de los tipos de diagrama, y se determinará el aporte y la validación para los objetivos PsiDiagram. En la Tabla 8-20 se muestra el proyecto ADGBase, y los tipos de diagrama ADG.

Tabla 8-20. Proyectos ADG de tipo PsiDiagram en PsiEnvironment.

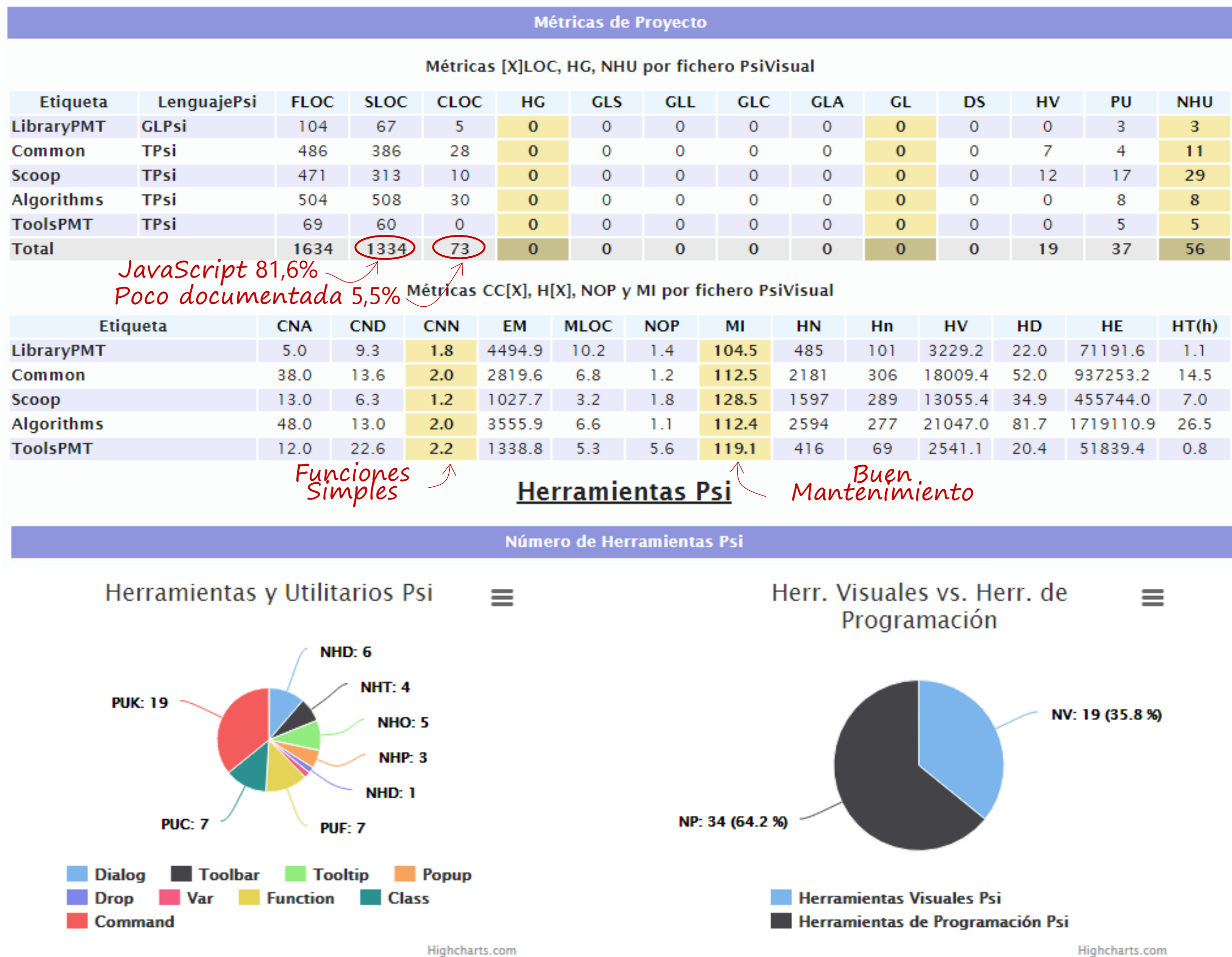
Caso de Estudio	Estilo	GLPsi	DPsi	TPsi	WPPsi	DFPsi	#LCPsi	Otros	Total
ADGBase (Proyecto ADG) *URL*: http://hilas.ii.uam.es/project?d=ADGBase	-	1 #104	0 #0	4 #1.530	0 #0	0 #0	5 #1.634	1	6 #1.693
ADG-ES (Modelo de estados) *URL*: http://hilas.ii.uam.es/project?d=ADG-ES	3	1 #102	2 #29	1 #282	1 #24	0 #0	5 #437	0 #0	5 #437
ADG-FU (Llamado de funciones SCOOP) *URL*: http://hilas.ii.uam.es/project?d=ADG-FU	1	1 93	2 #28	1 #205	1 #24	0 #0	5 #350	0 #0	5 #350
ADG-GA (Diagrama de GANTT) *URL*: http://hilas.ii.uam.es/project?d=ADG-GA	1	1 #217	2 #60	1 #394	1 #24	0 #0	5 #696	1 #27	6 #732
ADG-MD (Modelo de datos) *URL*: http://hilas.ii.uam.es/project?d=ADG-MD	3	1 #160	2 #63	1 #234	1 #24	0 #0	5 #481	0 #0	5 #481
ADG-MP (Mapa de objetos SCOOP) *URL*: http://hilas.ii.uam.es/project?d=ADG-MP	1	1 #84	2 #39	1 #203	1 #24	0 #0	5 #350	0 #0	5 #350
ADG-PE (Diagrama de PERT) *URL*: http://hilas.ii.uam.es/project?d=ADG-PE	2	1 #235	2 #84	1 #359	1 #24	0 #0	5 #702	0 #0	5 #702
ADG-RE (Diagrama conceptos-relaciones) *URL*: http://hilas.ii.uam.es/project?d=ADG-RE	4	1 #178	2 #79	1 #207	1 #24	0 #0	5 #488	0 #0	5 #488
ADG-RX (Diag. conceptos-relaciones extendido) *URL*: http://hilas.ii.uam.es/project?d=ADG-RX	1	1 #172	2 #79	1 #206	1 #24	0 #0	5 #481	0 #0	5 #481
ADG-WB (Diagrama de árbol WBS) *URL*: http://hilas.ii.uam.es/project?d=ADG-WB	1	1 #126	2 #40	1 #122	1 #24	0 #0	5 #312	0 #0	5 #312
Total número de ficheros **Total número de líneas**	**17**	**10** **#1.471**	**18** **#501**	**13** **#3.742**	**9** **#217**	**0** **#0**	**50** **#5.931**	**2** **#86**	**52** **#6.017**

Nota: **GLPsi\DPsi\TPsi\WPPsi\DFPsi**: programas Psi / # líneas Psi; **#Psi**: total programas Psi / # total líneas código Psi; **Otros**: Otros ficheros / # líneas; **Total**: #Psi+Otros. **Código fuente también disponible en DC adjunto.**

En el Proyecto ADG se crearon 52 ficheros, de los cuales el 19,2% (10) son librerías gráficas Psi, el 34,6% (18 ficheros) son diagramación Psi, el 25% (13) son herramientas Psi y el 17.3% (9) son marco de página y secciones Psi. Se escribieron un total de 5.938 líneas de código Psi, de las cuales el 27,5% de las líneas son del proyecto base ADGBase. Para ver el detalle de implementación ADGBase y sus métricas y de cada TypeADG y sus métricas, entrar al proyecto PsiDiagram correspondiente.

8.3.2.1 Análisis de métricas de ADGBase

En la Figura 8-17 se muestra el resumen de las métricas PsiDiagram para el proyecto ADGBase. Implementa 5 programas Psi, de los cuales el 80% son herramientas Psi, del total de líneas FLOC, el 81,6% son líneas de código JavaScript asociadas a la lógica de las diferentes herramientas y está poco documentado (CLOC 5,5%).



Métricas de Proyecto

Métricas [X]LOC, HG, NHU por fichero PsiVisual

Etiqueta	LenguajePsi	FLOC	SLOC	CLOC	HG	GLS	GLL	GLC	GLA	GL	DS	HV	PU	NHU
LibraryPMT	GLPsi	104	67	5	0	0	0	0	0	0	0	0	3	3
Common	TPsi	486	386	28	0	0	0	0	0	0	0	7	4	11
Scoop	TPsi	471	313	10	0	0	0	0	0	0	0	12	17	29
Algorithms	TPsi	504	508	30	0	0	0	0	0	0	0	0	8	8
ToolsPMT	TPsi	69	60	0	0	0	0	0	0	0	0	0	5	5
Total		1634	1334	73	0	0	0	0	0	0	0	19	37	56

JavaScript 81,6%
Poco documentada 5,5%

Métricas CC[X], H[X], NOP y MI por fichero PsiVisual

Etiqueta	CNA	CND	CNN	EM	MLOC	NOP	MI	HN	Hn	HV	HD	HE	HT(h)
LibraryPMT	5.0	9.3	1.8	4494.9	10.2	1.4	104.5	485	101	3229.2	22.0	71191.6	1.1
Common	38.0	13.6	2.0	2819.6	6.8	1.2	112.5	2181	306	18009.4	52.0	937253.2	14.5
Scoop	13.0	6.3	1.2	1027.7	3.2	1.8	128.5	1597	289	13055.4	34.9	455744.0	7.0
Algorithms	48.0	13.0	2.0	3555.9	6.6	1.1	112.4	2594	277	21047.0	81.7	1719110.9	26.5
ToolsPMT	12.0	22.6	2.2	1338.8	5.3	5.6	119.1	416	69	2541.1	20.4	51839.4	0.8

Figura 8-17. Resumen de métricas del proyecto ADGBase.

La complejidad ciclomática promedio CNN de las funciones/métodos del código JavaScript CLOC del proyecto ADGBase indica que son "funcionalidades simples" y tiene buena capacidad de mantenimiento MI. Es de notar que el tiempo de entendimiento del programa *Algorithms* (26,6 horas) requirió por ejemplo, un 21% de esfuerzo más que el programa *LibraryPMT* de 65 líneas de código JavaScript.

El proyecto ADGBase tiene un total de 56 herramientas de las cuales el 66% (34 herramientas en programas TPsi y 3 herramientas en programas GLPsi) son elementos de programación, y el resto 34% (19 herramientas) son herramientas visuales. Los diagramas circulares de la Figura 8-17 muestran el compendio de herramientas Psi, solamente de los programas TPsi. Los *comandos* representan el 35,8% (19 comandos) e implementan las funcionalidades de la mayoría de utilitarios de los diagramas generados, por ejemplo: redimensionamiento, impresión, escala, editar código Psi, depurador Psi, cambios de color, selecciones/deselecciones, bordes y puntos de chequeo, etc.

8.3.2.2 Análisis de métricas por tipo de diagrama ADG

En la Figura 8-18 se muestra el resumen de las métricas PsiDiagram para los tipos de diagrama de ADG. En promedio de escribieron 477 líneas FLOC, de las cuales en promedio, el 37,4% (179 líneas) son código JavaScript. Los tipos de diagrama que requirieron más líneas de código son los diagramas de PERT (702 líneas), y diagrama de GANTT (696 líneas), debido a su complejidad y detalle. Por otro lado, es de

notar las pocas líneas código JavaScript que se requieren para generar los diagramas: llamado de funciones SCOOP (82 líneas), mapa de objetos SCOOP (85 líneas) y diagramas de árbol WBS (80 líneas).

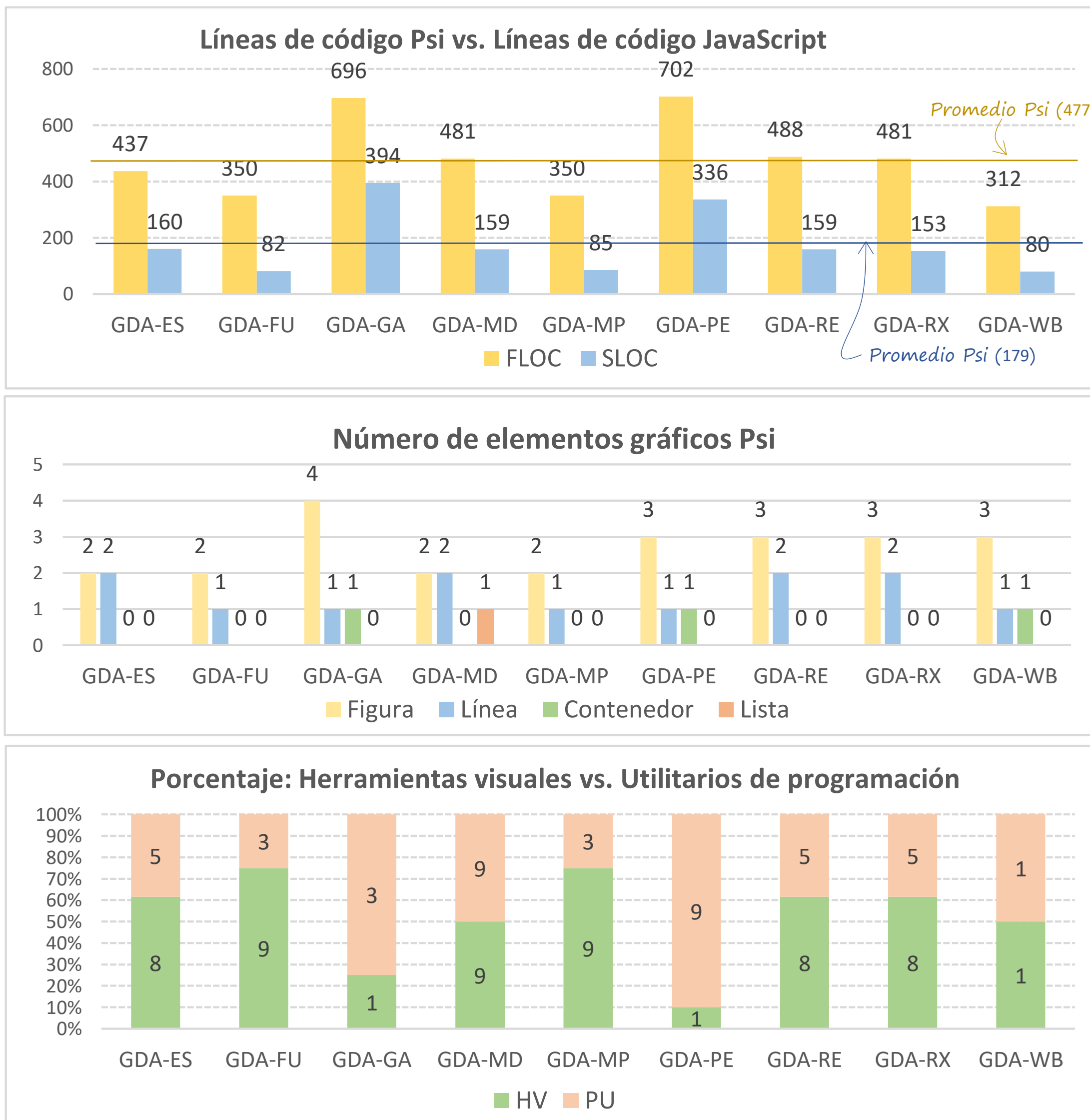


Figura 8-18. Resumen de métricas PsiDiagram para los proyectos de los tipos de diagrama ADG.

Por otro lado, el número de elementos gráficos total Psi (ver Figura 8-18) es de 41 elementos, de los cuales el 58,5% (24) son figuras y el 31,7% (13) son líneas, donde apenas el 9,8% (4) son 4 contenedores y una lista. El diagrama de GANTT es el que requiere más de elementos gráficos (6) y los que menos requieren son llamado de funciones SCOOP y mapa de objetos SCOOP. Pero es interesante ver el diagrama conceptos-relaciones, solo requirió de 5 elementos gráficos para generar 4 estilos diferentes de diagramas.

Por último, al analizar la comparativa de herramientas visuales HV versus elementos de programación PU (ver Figura 8-18), el promedio de HV es de 6 herramientas por diagramas, mientras que el promedio PU es aproximadamente 5 elementos. Los diagramas que más utilizan HV son llamado de funciones SCOOP, mapa de objetos SCOOP y Modelo de datos (con 9 herramientas); mientras que los diagramas que más utilizan PU son Diagrama de PERT y Modelo de datos (con 9 elementos).

8.3.2.3 Resumen VOG para el Proyecto ADG

En la Tabla 8-21 se muestra el resumen de características validadas por los proyectos ADG concernientes a los objetivos relacionados con PsiDiagram. Cabe notar que con estos proyectos se revalida nuevamente la totalidad de las funcionalidades para la creación de diagramas basados en PsiDiagram. Y se valida ampliamente el generador de diagramas de PsiDiagram.

Tabla 8-21. VOG: Resumen de características validadas en los proyectos ADG.

Nº	Características	Proyectos	
		ADGBase	Tipos ADG
7	Creación de diagramas basados en GLPsi y DPsi	F=3	S=24; L=13; C=3; A=1; F=7; E=42; M=4
8	Creación de herramientas asociadas con TPsi	HV (D=6; B=4; T=4; R=1; P=3). PU (V=1; F=7; C=7; K=19)	HV (D=18; B=15; T=1; R=7; M=6; P=1). PU(V=5; F=17; M=; C=1; K=3; L=3)
10	ADG y RD basados en PsiDiagram	-	G=9; R=1*

Nota: (*) prototipo de recomendación de diagramas conceptos-relaciones para determinar qué estilo recomendar.

8.3.3 PROYECTO SID: SCOOP INTERFACE DIAGRAM

En esta sección se resumirá el Proyecto SID (SCOOP Interface Diagram) que tiene como objetivo la creación y edición de diagramas en SCOOP, y los diferentes tipos de diagrama basados (ver sección 7.3). Se analizarán las métricas de tipo PsiDiagram tanto del proyecto SIDBase como de sus tipos de diagrama, y se determinará el aporte y la validación para los objetivos PsiDiagram. En la Tabla 8-22 se muestra el proyecto SIDBase, como los proyectos de cada tipo de diagrama SID.

Tabla 8-22. Proyectos SID de tipo PsiDiagram en PsiEnvironment.

Casos de Estudio	GLPsi	DPsi	TPsi	WPPsi	DFPsi	#LCPsi	Otros	Total
SIDBase (Proyecto SID) *URL*: http://hilas.ii.uam.es/project?d=SIDBase	1 #76	2 #44	7 #2.566	1 #24	0 #0	11 #2.710	2 #233	13 #2.943
SID-CR (Modelo de conceptos-relaciones para *Sketch*) *URL*: http://hilas.ii.uam.es/project?d=SID-CR	1 #104	2 #31	1 #154	0 #0	1 #20	5 #309	1 #27	6 #336
SID-SR (Modelo de estados para *Sketch*) *URL*: http://hilas.ii.uam.es/project?d=SID-SR	1 #88	2 #46	1 #205	0 #0	1 #11	5 #350	1 #26	6 #376
SID-PM (Modelo de procesos para *Sketch*) *URL*: http://hilas.ii.uam.es/project?d=SID-PM	1 #167	2 #58	2 #351	0 #0	1 #22	6 #598	1 #22	7 #598
SID-SCR (Modelo de conceptos-relaciones para *Modelo Técnico*) *URL*: http://hilas.ii.uam.es/project?d=SID-SCR	1 #133	2 #31	1 #160	0 #0	1 #20	5 #344	1 #31	6 #375
SID-SSR (Modelo de estados para *Modelo Técnico*) *URL*: http://hilas.ii.uam.es/project?d=SID-SSR	1 #87	2 #43	1 #205	0 #0	1 #17	5 #352	1 #26	6 #378
SID-SBPM (BPM para *BPMs y Eventos*) *URL*: http://hilas.ii.uam.es/project?d=SID-SBPM	1 #244	2 #58	1 #837	0 #0	1 #16	5 #1.155	1 #77	6 #1.232
Total número de ficheros **Total número de líneas**	**7** **#899**	**14** **#311**	**14** **#4.478**	**1** **#24**	**6** **#106**	**42** **#5.818**	**8** **#226**	**50** **#6.044**

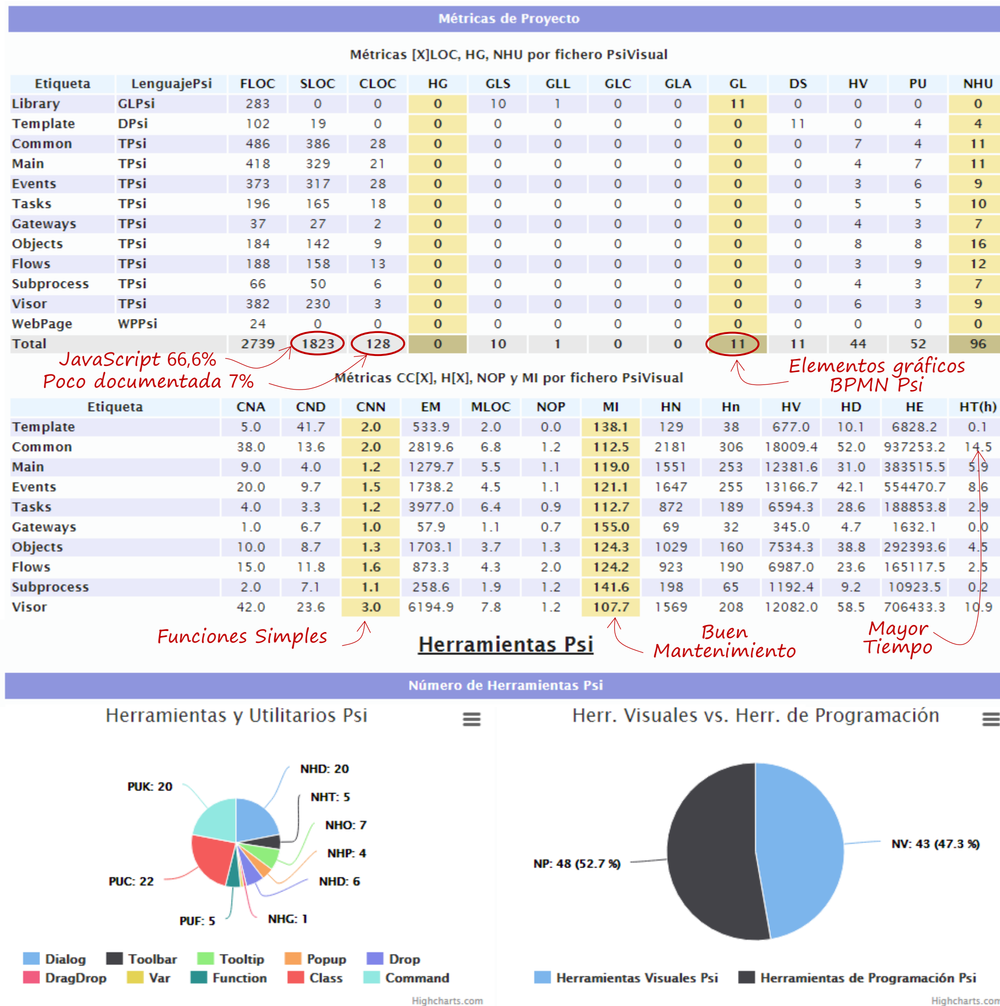
Nota: **GLPsi\DPsi\TPsi\WPPsi\DFPsi**: programas Psi / # líneas Psi; **#Psi**: total programas Psi / # total líneas código Psi; **Otros**: Otros ficheros / # líneas; **Total**: #Psi+Otros. **Código fuente también disponible en DC adjunto.**

En el Proyecto SID se crearon 50 ficheros, de los cuales el 14% (7) son librerías gráficas Psi, el 28% (14) son diagramación Psi, 28% (14) son herramientas Psi, el 2% (1) son marco de página y secciones Psi, y el resto 18,2% (14) son ficheros de recursos.

Se escribieron un total de 5.818 líneas de código Psi en los diferentes lenguajes GLPsi, DPsi, TPsi, WPPsi y DFPsi, de las cuales el 46,6% de las líneas son del proyecto base SIDBase. Edite el proyecto PsiDiagram respectivo para ver el detalle de implementación de SIDBase y de cada proyecto TypeSID y sus métricas.

8.3.3.1 Análisis de métricas de SIDBase

En la Figura 8-19 se muestra el resumen de las métricas PsiDiagram para el proyecto SIDBase. Implementa 11 programas Psi, de los cuales el 63,6% son herramientas Psi, del total de líneas FLOC, el 74,3% son líneas de código JavaScript asociadas a la lógica de Proyecto SID, pero su documentación es escasa (CLOC 5%).



Métricas de Proyecto

Métricas [X]LOC, HG, NHU por fichero PsiVisual

Etiqueta	LenguajePsi	FLOC	SLOC	CLOC	HG	GLS	GLL	GLC	GLA	GL	DS	HV	PU	NHU
Library	GLPsi	283	0	0	0	10	1	0	0	11	0	0	0	0
Template	DPsi	102	19	0	0	0	0	0	0	0	11	0	4	4
Common	TPsi	486	386	28	0	0	0	0	0	0	0	7	4	11
Main	TPsi	418	329	21	0	0	0	0	0	0	0	4	7	11
Events	TPsi	373	317	28	0	0	0	0	0	0	0	3	6	9
Tasks	TPsi	196	165	18	0	0	0	0	0	0	0	5	5	10
Gateways	TPsi	37	27	2	0	0	0	0	0	0	0	4	3	7
Objects	TPsi	184	142	9	0	0	0	0	0	0	0	8	8	16
Flows	TPsi	188	158	13	0	0	0	0	0	0	0	3	9	12
Subprocess	TPsi	66	50	6	0	0	0	0	0	0	0	4	3	7
Visor	TPsi	382	230	3	0	0	0	0	0	0	0	6	3	9
WebPage	WPPsi	24	0	0	0	0	0	0	0	0	0	0	0	0
Total		2739	1823	128	0	10	1	0	0	11	11	44	52	96

Métricas CC[X], H[X], NOP y MI por fichero PsiVisual

Etiqueta	CNA	CND	CNN	EM	MLOC	NOP	MI	HN	Hn	HV	HD	HE	HT(h)
Template	5.0	41.7	2.0	533.9	2.0	0.0	138.1	129	38	677.0	10.1	6828.2	0.1
Common	38.0	13.6	2.0	2819.6	6.8	1.2	112.5	2181	306	18009.4	52.0	937253.2	14.5
Main	9.0	4.0	1.2	1279.7	5.5	1.1	119.0	1551	253	12381.6	31.0	383515.5	5.9
Events	20.0	9.7	1.5	1738.2	4.5	1.1	121.1	1647	255	13166.7	42.1	554470.7	8.6
Tasks	4.0	3.3	1.2	3977.0	6.4	0.9	112.7	872	189	6594.3	28.6	188853.8	2.9
Gateways	1.0	6.7	1.0	57.9	1.1	0.7	155.0	69	32	345.0	4.7	1632.1	0.0
Objects	10.0	8.7	1.3	1703.1	3.7	1.3	124.3	1029	160	7534.3	38.8	292393.6	4.5
Flows	15.0	11.8	1.6	873.3	4.3	2.0	124.2	923	190	6987.0	23.6	165117.5	2.5
Subprocess	2.0	7.1	1.1	258.6	1.9	1.2	141.6	198	65	1192.4	9.2	10923.5	0.2
Visor	42.0	23.6	3.0	6194.9	7.8	1.2	107.7	1569	208	12082.0	58.5	706433.3	10.9

Figura 8-19. Resumen de métricas del proyecto SIDBase.

La complejidad ciclomática promedio CNN de las funciones/métodos del código JavaScript CLOC del proyecto SIDBase indica que son "funcionalidades simples" y tiene buena capacidad de mantenimiento MI con un promedio de 119 en los diferentes programas Psi. Es de notar que el 60,9% del esfuerzo de implementación es para los programas TPsi *Tools*, *Node* y *Connectors*, pero el 51% del tiempo HT(h) es para los programas TPsi *Main* y *Tools*.

El proyecto SIDBase tiene un total de 99 herramientas de las cuales el 64,6% (61 son herramientas en programas TPsi y 3 herramientas en programas GLPsi) son elementos de programación, y el resto 35,4% (35) son herramientas visuales. Los diagramas circulares de la Figura 8-19 muestran el compendio de herramientas Psi de los programas TPsi.

Los *comandos* (35), *funciones* (13) y *clases* (9) que representa el 57,6% del proyecto, implementan las operaciones genéricas de creación, edición, eliminación, consulta y configuración de los elementos

gráficos (nodos y conectores) de los diferentes diagramas. Se crearon un conjunto de funciones nativas entre el diagrama y SCOOP que dieran soporte a las diferentes operaciones. Además, se implementó el lenguaje DFPsi, que aporta la creación dinámica de formularios en cajas de diálogo, para la creación y edición de la información asociada a los elementos gráficos.

Adicionalmente se crearon dos tipos de artefactos genéricos: un *agrupador* visual de elementos gráficos y un *anotador* de comentarios sobre nodos Psi. Sus programas son: *ArtifactsGL*, *ArtifactsD* y *ArtifactsT*.

8.3.3.2 Análisis de métricas por tipo de diagrama SID

En la Figura 8-20 se muestra el resumen de las métricas PsiDiagram para los tipos de diagrama de SID. En promedio de escribieron 518 líneas FLOC, de las cuales el 37% (193) son código JavaScript. Además, el 75% de código SLOC fue escrito para los proyectos SID-SBPM y SID-PM, y es de esperarse. Esta cifra es comprensible dado que el proyecto SID-SBPM (646 SLOC) incorpora la generación automática de BPM.

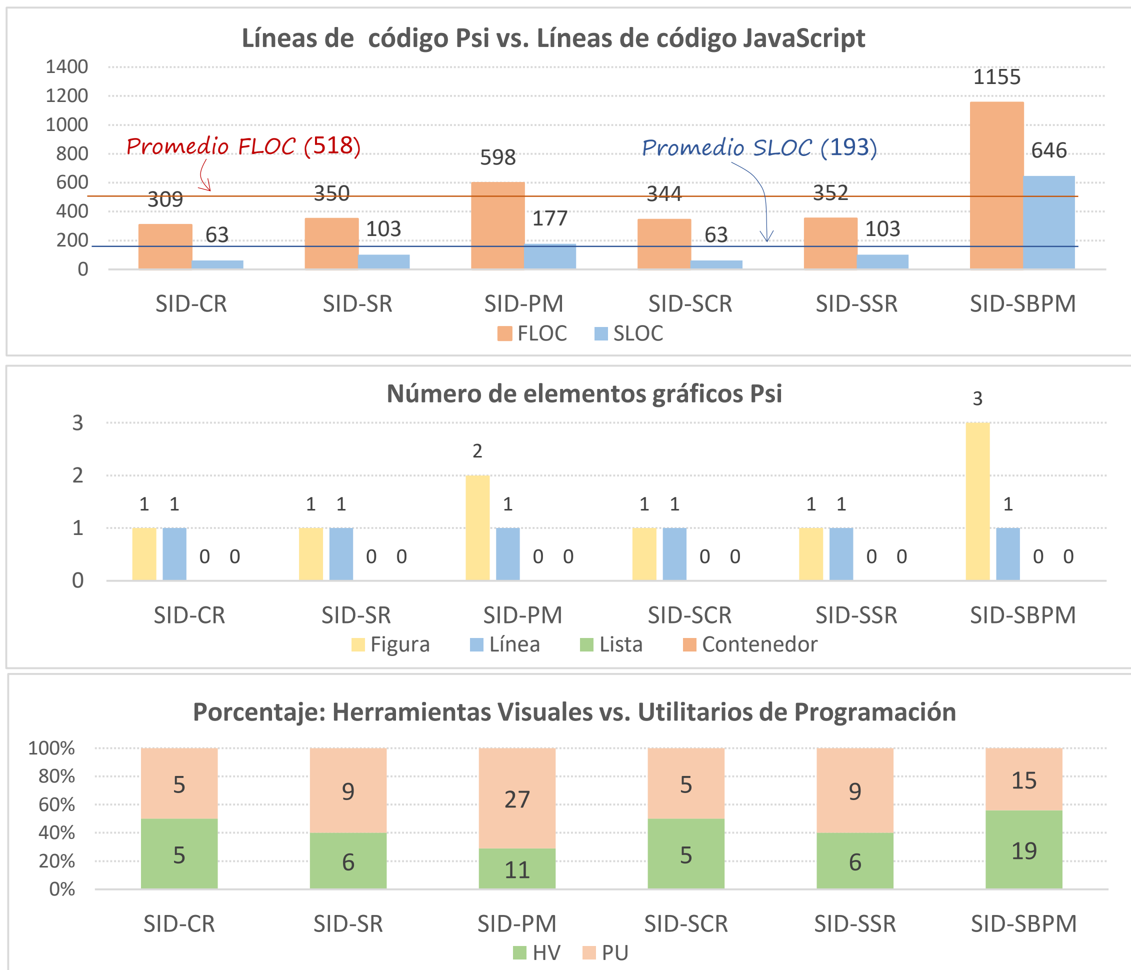


Figura 8-20. Resumen de métricas PsiDiagram para los proyectos de los tipos de diagrama SID.

Los tipos de diagrama que requirieron más líneas de código son el diagrama BPM (1.139) y modelo de procesos (586). Por otro lado, es de notar las pocas líneas de código JavaScript que se requieren para crear los diagramas: modelos de conceptos-relaciones (63) y modelo de estados (103).

Por otro lado, el número de elementos gráficos total Psi (ver Figura 8-20) es de 15 elementos, de los cuales el 60% (9) son figuras y el 40% (6) son líneas, y no han requerido hasta el momento contenedores y listas.

Por último, al analizar la comparativa de herramientas visuales HV versus elementos de programación PU (ver Figura 8-20), el promedio de HV es de 12 herramientas por diagramas, mientras que el promedio PU es aproximadamente 20 elementos. El diagrama que más utiliza HV es modelo de procesos (con 27 herramientas); mientras que el diagrama que más utiliza PU es el modelo de procesos (con 19 elementos).

8.3.3.3 Resumen VOG para el Proyecto SID

En la Tabla 8-23 se muestra el resumen de características validadas por los proyectos SID concernientes a los objetivos relacionados con PsiDiagram. Es de notar que en la diagramación Psi no se tiene información, dado que estas funcionalidades se han validado con los diagramas de clases y diagramas PsiLSD.

Tabla 8-23. VOG: Resumen de características validadas en el Proyecto SID.

Nº	Características	Proyectos	
		SIDBase	Tipos SID
7	Creación de diagramas basados en GLPsi y DPsi	S=2; L=1; F=3; E=3	E=26
8	Creación de herramientas asociadas con TPsi	HV (D=10; B=4; T=5; G=2; R=4; P=9). PU (V=1; F=13; C=9; K=35)	HV (D=1; B=13; T=4; R=17; M=6; P=11). PU(V=8; F=32; M=; C=1; K=24)
10	Generación de diagramas y recomendación de diagramas	-	G=6

8.3.4 PROYECTO BPME Y SMAPPS

En esta sección se resumirá el Proyecto **BPMEBase** que tiene como objetivo la creación y edición de procesos de negocios basado en la especificación BPMN 2.0 (OMG, 2011) en SCOOP (forma parte del proyecto BPME de SCOOP, ver sección 7.4). Se analizarán las métricas de tipo PsiDiagram tanto del proyecto BPMEBase como sus proyectos adjuntos, además se determinará el aporte a la validación para los objetivos PsiDiagram. En la Tabla 8-24 se muestra el proyecto BPMEBase, como los proyectos adjuntos. Como proyecto adjunto se incluye el Proyecto **BPMSMAPPS**, proyecto para la *definición de servicios SMAPPS* (ver sección 7.5).

Tabla 8-24. Proyecto BPMEBase y proyectos adjuntos BPME.

Casos de Estudio	GLPsi	DPsi	TPsi	WPPsi	DFPsi	#LCPsi	Otros	Total
BPMEBase (Proyecto BPMN 2.0) *URL*: http://hilas.ii.uam.es/project?d=BPMEBase	1 #283	1 #102	9 #2.330	1 #24	0 #0	12 #2.739	1 #405	13 #3.144
BPMVisor (Visor de proyectos BPME) *URL*: http://hilas.ii.uam.es/project?d=BPMEVisor	0 #0	1 #13	1 #190	0 #0	1 #20	3 #223	5 #404	8 #627
BPMSCOOP (Generación automática de diagramas BPM para *BPMs y Eventos*) *URL*: http://hilas.ii.uam.es/project?d=BPMESCOOP	0 #0	0 #0	3 #1.421	0 #0	1 #21	4 #1.442	1 #90	5 #1.532
BPMSMAPPS (Definición de servicios SMAPPS) *URL*: http://hilas.ii.uam.es/project?d=BPMESMAPPS	0 #0	0 #0	1 #168	0 #0	1 20	2 #188	1 #114	3 #302
Total número de ficheros **Total número de líneas**	**1** **#283**	**2** **#115**	**14** **#4.109**	**1** **#24**	**3** **#61**	**21** **#4.592**	**8** **#1.013**	**28** **#5.605**

Nota: **GLPsi\DPsi\TPsi\WPPsi\DFPsi**: programas Psi / # líneas Psi; **#Psi**: total programas Psi / # total líneas código Psi; **Otros**: Otros ficheros / # líneas; **Total**: #Psi+Otros. **Código fuente también disponible en DC adjunto.**

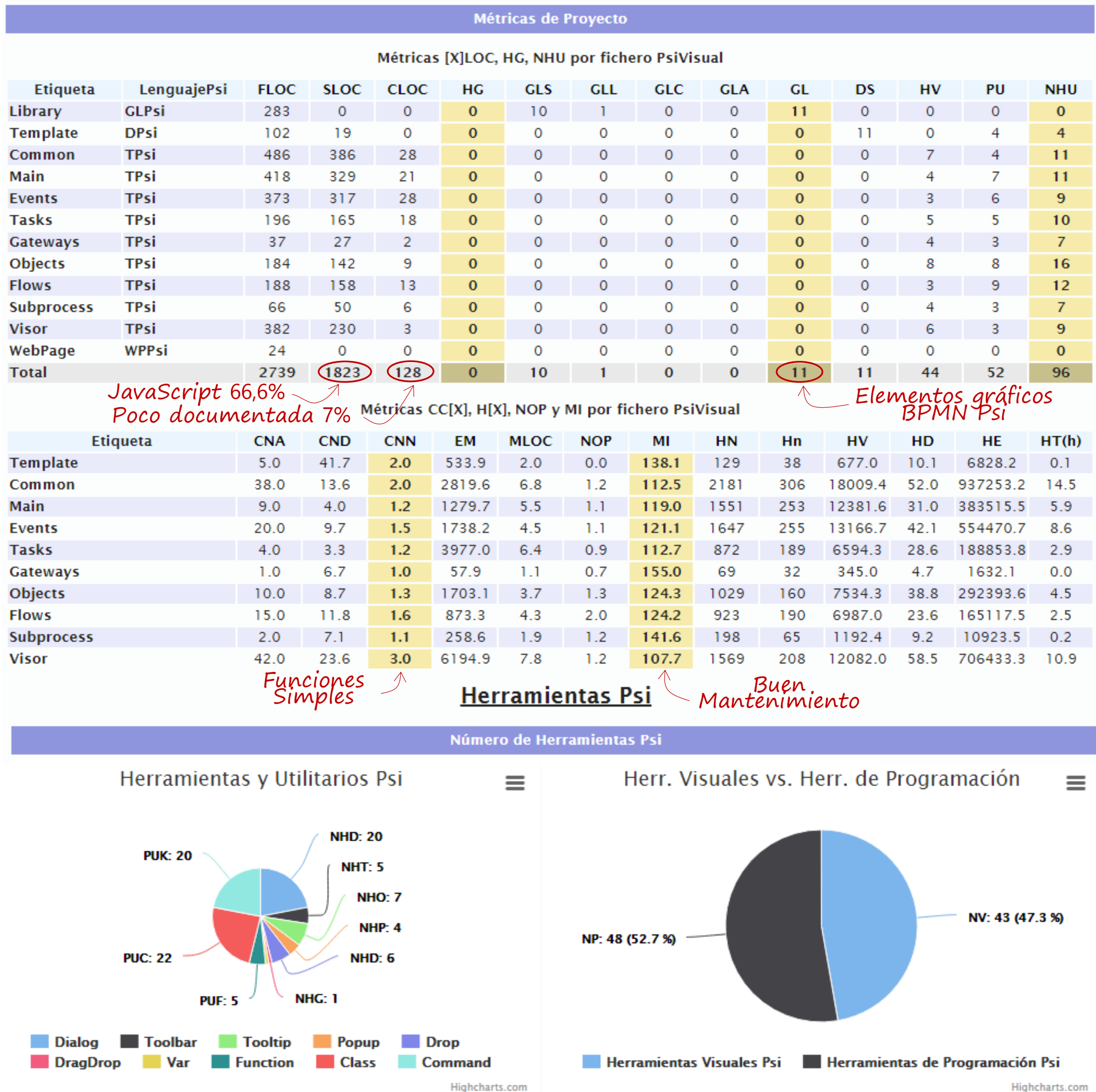
En el Proyecto de la Tabla 8-24 se crearon 28 ficheros, de los cuales el 50% (14) son herramientas Psi, el 25% (7) son ficheros de configuración y recursos, y el 25% (7) restante son programas GLPsi, DPsi,

WPPsi y DFPsi.

Se escribieron un total de 4.592 líneas de código Psi, de las cuales el 59,6% de las líneas son del proyecto base BPMEBase. Edite el proyecto PsiDiagram respectivo para ver el detalle de implementación BPMEBase y de los proyectos adjuntos, y sus métricas.

8.3.4.1 Métricas de BPMEBase

En la Figura 8-21 se muestra el resumen de las métricas PsiDiagram para el proyecto BPMEBase. Implementa 12 programas Psi, de los cuales el 75% son herramientas Psi, del total de líneas FLOC (2.739 líneas), el 66,6% son líneas de código JavaScript asociadas a la lógica para la administración de BPMN 2.0, pero su documentación es muy escasa (CLOC 7%).



Métricas de Proyecto

Métricas [X]LOC, HG, NHU por fichero PsiVisual

Etiqueta	LenguajePsi	FLOC	SLOC	CLOC	HG	GLS	GLL	GLC	GLA	GL	DS	HV	PU	NHU
Library	GLPsi	283	0	0	0	10	1	0	0	11	0	0	0	0
Template	DPsi	102	19	0	0	0	0	0	0	0	11	0	4	4
Common	TPsi	486	386	28	0	0	0	0	0	0	0	7	4	11
Main	TPsi	418	329	21	0	0	0	0	0	0	0	4	7	11
Events	TPsi	373	317	28	0	0	0	0	0	0	0	3	6	9
Tasks	TPsi	196	165	18	0	0	0	0	0	0	0	5	5	10
Gateways	TPsi	37	27	2	0	0	0	0	0	0	0	4	3	7
Objects	TPsi	184	142	9	0	0	0	0	0	0	0	8	8	16
Flows	TPsi	188	158	13	0	0	0	0	0	0	0	3	9	12
Subprocess	TPsi	66	50	6	0	0	0	0	0	0	0	4	3	7
Visor	TPsi	382	230	3	0	0	0	0	0	0	0	6	3	9
WebPage	WPPsi	24	0	0	0	0	0	0	0	0	0	0	0	0
Total		2739	1823	128	0	10	1	0	0	11	11	44	52	96

Métricas CC[X], H[X], NOP y MI por fichero PsiVisual

Etiqueta	CNA	CND	CNN	EM	MLOC	NOP	MI	HN	Hn	HV	HD	HE	HT(h)
Template	5.0	41.7	2.0	533.9	2.0	0.0	138.1	129	38	677.0	10.1	6828.2	0.1
Common	38.0	13.6	2.0	2819.6	6.8	1.2	112.5	2181	306	18009.4	52.0	937253.2	14.5
Main	9.0	4.0	1.2	1279.7	5.5	1.1	119.0	1551	253	12381.6	31.0	383515.5	5.9
Events	20.0	9.7	1.5	1738.2	4.5	1.1	121.1	1647	255	13166.7	42.1	554470.7	8.6
Tasks	4.0	3.3	1.2	3977.0	6.4	0.9	112.7	872	189	6594.3	28.6	188853.8	2.9
Gateways	1.0	6.7	1.0	57.9	1.1	0.7	155.0	69	32	345.0	4.7	1632.1	0.0
Objects	10.0	8.7	1.3	1703.1	3.7	1.3	124.3	1029	160	7534.3	38.8	292393.6	4.5
Flows	15.0	11.8	1.6	873.3	4.3	2.0	124.2	923	190	6987.0	23.6	165117.5	2.5
Subprocess	2.0	7.1	1.1	258.6	1.9	1.2	141.6	198	65	1192.4	9.2	10923.5	0.2
Visor	42.0	23.6	3.0	6194.9	7.8	1.2	107.7	1569	208	12082.0	58.5	706433.3	10.9

Figura 8-21. Resumen de métricas del proyecto BPMEBase.

La complejidad ciclomática promedio CNN de las funciones/métodos del código JavaScript CLOC del proyecto BPMEBase apunta que son “funcionalidades simples” y con buena capacidad de mantenimiento MI con un promedio de 126 en los diferentes programas Psi. Es de notar que el 46,4% del esfuerzo EM es para los programa TPsi *Visor* y *Common*, pero el 57,8% del esfuerzo HE es para los programas TPsi *Common*, *Events* y *Main* y el 50% aproximadamente de tiempo HT(h) es para los programas *Visor* y *Common*.

El proyecto BPMEBase tiene un total de 96 herramientas de las cuales el 54,2% (48 son herramientas en programas TPsi y 4 herramientas en programas DPsi) son elementos de programación, y el resto 45,8% (44) son herramientas visuales. Nuevamente, los diagramas circulares de la Figura 8-21 muestran el compendio de herramientas Psi de los programas TPsi.

Los *comandos* (20), *clases* (22) y *funciones* (5) que representa el 49% de herramientas del proyecto, implementan las operaciones de creación, edición, eliminación y configuración de los elementos gráficos del BPMN 2.0. Se creó la clase *BaseElement* que tiene la funcionalidad abstracta para el manejo de cualquier elemento del BPMN y se creó, además, los ficheros de configuración para determinar cuáles son los elementos gráficos disponibles en el diagrama.

8.3.4.2 Resumen VOG para el Proyecto BPME

En la Tabla 8-25 se muestra el resumen de características validadas por los proyectos BPMEBase y proyectos adjuntos concernientes a los objetivos relacionados con PsiDiagram. La validación relevante obtenida es la creación de un DSVL basado en PsiDiagram (Proyracto DSVLSMAPPS).

Tabla 8-25. VOG: Resumen de características validadas en Proyecto BPME.

Nº	Características	Proyectos	
		SIDBase	Adjuntos
7	Creación de diagramas basados en GLPsi y DPsi	S=10; L=1; F=4; E=11	S=1
8	Creación de herramientas asociadas con TPsi	HV (D=20; B=5; T=4; G=1; R=6; P=4). PU (V=1; F=5; C=22; K=20)	HV (M=6; P=19). PU(F=5; K=5; L=2)
9	Creación de DSVLs basados en PsiDiagram	-	DSVL=1
10	Generación de diagramas y recomendación de diagramas	-	G=4

8.3.5 NOTAS FINALES DE VALIDACIÓN

De acuerdo con la metodología cualitativa multicaso sugerida en (Yin, 2014; Baxter & Jack, 2008) para validar los objetivos de investigación de la tesis, se desarrollaron cuatro proyectos PsiDiagram para la creación y generación automática de diagramas y creación de DSVL:

i. **PsiBase**: creación y edición de diagramas de clases y diagramas PsiLSD.
ii. **ADGBase**: generación automática de diagramas ADG.
iii. **SIDBase**: creación y edición de diagramas en SCOOP.
iv. **BPMEBase**: creación y edición de procesos de negocios basado en la especificación BPMN 2.0.

De estos 4 proyectos se desprenden 21 proyectos PsiDiagram repartidos: PsiBase (3 proyectos con 13 diagramas de clases, 13 diagramas de clase básico y 17 diagramas PsiLSD), ADGBase (9 proyectos, generación de 8 tipos de diagrama), SIDBase (6 proyectos, creación y edición de 4 tipos de diagramas) y BPMEBase (3 proyectos, 2 proyectos creación y edición de BPMN y un proyecto de creación y edición de DSVL).

En resumen, un total de 25 proyectos PsiDiagram que validan las características de PsiDiagram (ver Tabla 8-26) enumerados VOG (Tabla 8-1). Además, los proyectos PsiDiagram revalidan nuevamente

todas las características de PsiEngine y PsiModel, ya validadas en la sección anterior.

Tabla 8-26. VOG: Número de validaciones de las características para PsiDiagram.

	Nº	Características	Número de Validaciones
PsiDiagram	7	Creación de diagramas basados en GLPsi	S=42; L=19; C=5; A=4; F=18; E=117; M=4; S*=478; L*=904; C*=42; A*=248
	8	Creación de herramientas asociadas a diagramas con TPsi	HV (D=69; B=44; T=21; F=1; G=3; R=35; M=19; P=49). PU (V=15; F=83; C=41; K=114; L=10)
	9	Creación de DSVLs basados en PsiDiagram	DSVL=1
	10	Generación de diagramas y recomendación de diagramas	G=19; R=1**

Nota: (*) validación del uso de elementos gráficos Psi en la diagramación Psi. (**) Se tiene un prototipo de recomendación.

8.4 RESUMEN DE RESULTADOS

Retomando el diagrama de bloques del Motor de Evaluación Psi de la Figura 3-1 (sección 3.1), en la Figura 8-22 se muestra el resumen de resultados para la validación de objetivos O1, O2, O3 y O4 de esta tesis. Se han diseñado e implementado 16 lenguajes Psi, más de 45.000 líneas de código Psi, más de 162 ficheros de recursos, para crear 12 aplicaciones web y un motor de aplicaciones móviles.

Para validar las características de **PsiEngine** del motor de evaluación de programas escritos en DSL con gramática XML en un CSWE (O1), se analizaron 18 casos de estudio en este documento, entre los que se destacan: PsiEnvironment, DSVL3 de SCOOP, BPMEEditor, BPMVisor, BPMSMAPPS, PsiEditDiagram, ADGTest, SIDTest, MiniPsi, FeedPsi, Anisha y Graphs. En los diferentes casos, el *Intérprete PsiXML* evalúa programas Psi de 16 lenguajes Psi diferentes para aplicaciones web y aplicaciones móviles.

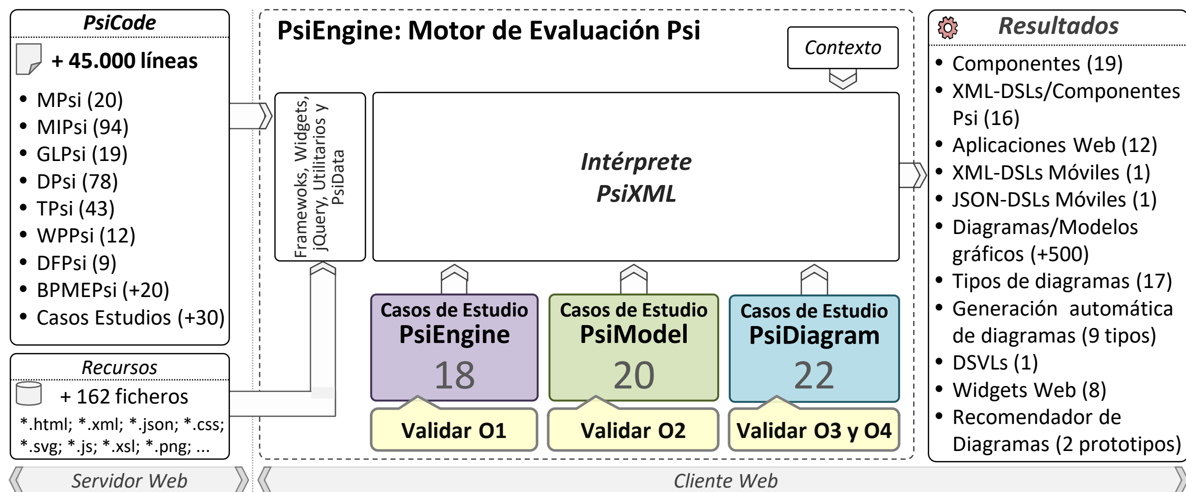


Figura 8-22. Resumen de resultados para la validación de objetivos O1, O2, O3 y O4.

Para validar las características de **PsiModel** del modelo de programación para la creación de DSL con gramática XML en un CSWE (O2), se llevaron a cabo 20 casos de estudio, donde se crearon 114 programas Psi (20 programas en lenguaje de Especificación MPsi, y 94 programas en lenguaje de Implementación MIPsi) y 114 ficheros de recursos, para crear el *Intérprete PsiXML*, 16 componentes Psi, 18 componentes JavaScript y 8 widgets web, para las diferentes aplicaciones web y aplicaciones móviles.

Para validar las características de **PsiDiagram** del diagrama programable que facilita el diseño e implementación de DSVL, diagramas, generación de diagramas y mecanismos de recomendación de diagramas en un CSWE, objetivo (O3), se abordaron y analizaron 22 casos de estudio, donde se crearon

152 programas Psi (entre lenguajes GLPsi, DPsi, TPsi y WPPsi) para la elaboración de 17 tipos de diagramas, 12 tipos de diagramas automáticos y un DSVL basado en BPMN 2.0, con ellos se han creado hasta el momento más de 500 diagramas aproximadamente. Para validar los mecanismos de recomendación de diagramas en un CSWE, objetivo (O4), se crearón dos prototipos.

En la Tabla 8-27 se muestra el resumen con el número de validaciones para cada una de las características del objetivo general de esta tesis.

Tabla 8-27. VOG: Número de validaciones de las características para PsiEngine, PsiModel y PsiModel.

	Nº	Características	Número de Validaciones
PsiEngine	1	Ejecución de programas Psi: (S) simple-simple (un programa de un lenguaje Psi); (P) múltiple-simple (múltiples programas de un lenguaje Psi); (M) múltiple-múltiple (múltiples programas escritos en múltiples lenguajes Psi)	S =1; P=11; M=8
	2	Asociación de información heterogénea: (X) XML; (J) JSON	X=25; J=10
	3	Compartir objetos e información entre: (S) diferentes programas escritos en el mismo lenguaje Psi; (M) diferentes programas escritos en diferentes lenguajes Psi	S=8; M=13
	4	Generación dinámica de una aplicación web por un programa Psi	3
PsiMode	5	Creación de elementos de programación usados en PsiModel: (V) variable; (F) función; (E) tipo enumerado; (O) Objeto; (C) Clase; (W) Web widget;	V=13;F=13;E=21;O=46; C=419; W=10
	6	Construcción de componentes usados en PsiModel: (C) crear componente JavaScript; (P) crear componentes Psi (es decir, crear lenguajes XML-DSLs)	C=19; P=16;
PsiDiagram	7	Creación de diagramas basados en GLPsi: (S) figura; (L): línea; (C) contenedor; (A) lista; (K) comandos; (F) funciones; (E) Skeleton; (M) modificador gráfico	S=42; L=19; C=5; A=4; F=18; E=117; M=4; S*=478; L*=904; C*=42; A*=248
	8	Creación de herramientas asociadas a diagramas con TPsi. *Herramientas Visuales* HV: (D) caja de diálogo; (B) barra de herramientas; (T) ventana emergente; (F) marco de contenido; (G) arrastrar / soltar; (R) soltar; (M) menú; (P) menú flotante. *Utilidades de Programación* PU: (V) variable; (F) función; (M) modificador gráfico; (C) Clases; (K) Comando Psi; (L) carga de documento	HV (D=69; B=44; T=21; F=1; G=3; R=35; M=19; P=49). PU (V=15; F=83; C=41; K=114; L=10)
	9	Creación de DSVLs basados en PsiDiagram	DSVL=1
	10	(G) Generación automática de diagramas basados en PsiDiagram y (R) creación de mecanismos de recomendación de diagramas basados en la generación automática de PsiDiagram	G=19; R=1**

Nota: (*) validación del uso de elementos gráficos Psi en la diagramación Psi. (**) Se tiene un prototipo de recomendación.

En la Tabla 8-27 se han unido las validaciones de objetivos de PsiEngine y PsiModel mostradas en la Tabla 8-16, con las validaciones de los objetivos de PsiModel mostradas en la Tabla 8-26. Y se han adicionado las validaciones generadas por los diferentes proyectos PsiDiagram.

CONCLUSIÓN:

La metodología de validación realizada, ha sido la *metodología cualitativa de multicasos de estudio* sugerida por (Yin, 2014) y adaptada de la Ingeniería de Software en (Baxter & Jack, 2008). Con los casos de estudio resumidos en la Figura 8-22 y el resumen VOG de la Tabla 8-27, se cubren los aspectos relevantes de cada uno de los objetivos de investigación (O1), (O2), (O3) y (O4). En consecuencia puede darse por validado el Objetivo General a través de las propuestas planteadas las cuales permiten la especificación, creación y evaluación de XML-DSLs con capacidad de asociar información heterogénea (XML/JSON) en un cliente web, para la construcción de componentes, widgets, XML-DSLs, DSVLs, modelos gráficos y diagramas que ayuden a dar soluciones a problemas específicos.

Capítulo 9. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se presentan las conclusiones y los trabajos futuros para los objetivos planteados en esta tesis. Se presentan las conclusiones en relación al objetivo general y los objetivos específicos (sección 9.1) y se listan los resultados alcanzados (sección 9.2). Se revisa el cumplimiento de las hipótesis que se plantearon (sección 9.3), así como algunos trabajos futuros pendientes de realizar a la conclusión de esta memoria (sección 9.4). Por último, se expresan algunas reflexiones finales (sección 9.5).

9.1 INTRODUCCIÓN

Los DSL son lenguajes de programación que ofrecen abstracciones de alto nivel para facilitar el diseño, la definición y/o la implementación de sistemas de información que dan solución a problemas de un dominio específico. Si los DSLs se definen con gramática XML (XML-DSL), permite que estos sean fáciles de usar, extensibles y combinables, y se puedan usar en diferentes plataformas y sistemas de información.

En un Entorno Cliente Web CSWE (conjunto de tecnologías, lenguajes, metodologías de programación y bibliotecas disponibles en el cliente web para la implementación de componentes y/o widgets), las propuestas de creación de XML-DSLs son bastante escasas, tal y como se documentó en el Capítulo 2. En este contexto, el trabajo presentado en esta tesis se ha orientado hacia la implementación de un *intérprete* de XML-DSLs en JavaScript que evalúa múltiples programas para múltiples lenguajes en cualquier navegador y dispositivo, con capacidad de coexistencia e intercambio de información con otros programas, y con la capacidad de administrar múltiples fuentes heterogéneas de datos XML/JSON de forma nativa. Gracias a este intérprete hemos podido aportar un valor añadido al desarrollo de soluciones a problemas en un entorno empresarial.

Por otra parte, un DSVL es un DSL que se define mediante elementos gráficos que facilita el desarrollo de nuevos sistemas de información, incluyendo a usuarios que, no siendo programadores pero si expertos en el dominio del problema. Hasta el presente la mayor parte de las propuestas de DSVLs se han dirigido a resolver problemas en aplicaciones de escritorio y en entornos de desarrollo integrado (IDE). En un CSWE, la mayoría de aplicaciones web y frameworks están orientadas a la elaboración de diagramas y modelos gráficos y a la visualización de información heterogénea en línea. Es decir, no hay entornos de modelado visual o herramientas visuales para la creación y evaluación de DSVLs basadas en las tecnologías de un cliente web. En este marco de referencia y basándonos en el motor implementado, nuestra propuesta se concreta en el concepto de *diagrama programable* para definir librerías gráficas generadas a partir de SVG y serializar modelos gráficos y DSVLs con gramática XML sobre un CSWE. Este diagrama tiene la capacidad de asociar fuentes heterogéneas de información

XML/JSON; añadir herramientas visuales como cajas de diálogo, barras de herramientas, ventanas emergentes, menús, etc.; usar elementos básicos de programación como variables, funciones y clases; asociar herramientas de generación automática de diagramas; y proponer mecanismos de recomendación de diagramas.

En la sección siguiente vamos a detallar la lista de contribuciones que se derivan de la presente tesis.

9.2 CONTRIBUCIONES

Esta sección tiene como objetivo resumir las contribuciones aportadas por esta tesis. Como paso inicial recordemos el objetivo general:

> La especificación, creación y evaluación de XML-DSLs con capacidad de asociar información heterogénea (XML/JSON) en un cliente web, para la construcción de componentes, widgets, XML-DSLs, DSVLs, modelos gráficos y diagramas que ayuden a dar soluciones a problemas específicos.

Para evaluar la consecución de este objetivo general, se plantearon y abordaron los siguientes objetivos específicos de investigación:

- O1. Definir y desarrollar un motor de evaluación de programas escritos en XML-DSL con capacidad de asociar información heterogénea (XML/JSON) en un CSWE.
- O2. Definir y desarrollar un modelo de programación para la creación de XML-DSLs en un CSWE.
- O3. Definir y desarrollar una propuesta que facilite el diseño e implementación de diagramas programables para la creación de DSVLs y la generación automática de diagramas en un CSWE.
- O4. Crear mecanismos de recomendación gráfica a partir de la generación automática de diagramas del O3.

La Figura 9-1 facilita la comprensión y estructura de contenidos tratados en los diferentes capítulos para cubrir el objetivo general de esta misma, y ubica las contribuciones aportadas por los objetivos indicados específicos de investigación.

C1. PsiEngine: Motor de Evaluación Psi con capacidad de asociar información heterogénea (XML/JSON) para definir y evaluar XML-DSL en un CSWE

La creación y evaluación de XML-DSLs en un CSWE requiere de la implementación de un *intérprete especializado* para definir la gramática XML del lenguaje y la funcionalidad de cada etiqueta. En la Figura 9-1 se muestra el Motor de Evaluación Psi, denotado por **PsiEngine** (**P**rogrammable **S**olutions **I**nterpreter **Engine**), el cual combina los lenguajes HTML5, CSS3, JavaScript y DOM, con las tecnologías, servicios y herramientas de la Web 2.0, y el *intérprete especializado* (denominado *Intérprete PsiXML*), para la construcción de componentes web, widgets web y/o páginas web dinámicas que dan solución a problemas específicos dados en una aplicación web, o en partes de ella.

El *Intérprete PsiXML* es un framework JavaScript liviano y genérico (funciona en cualquier navegador de cualquier dispositivo) para interpretar y evaluar programas Psi basados en lenguajes Psi. Un *lenguaje Psi* es un XML-DSL que tiene asociada la estructura de la gramática XML del lenguaje (árbol de etiquetas) y la funcionalidad para cada etiqueta implementada en una clase JavaScript. Además, un lenguaje Psi hereda la funcionalidad de asociar información heterogénea en formato XML y/o JSON de

forma nativa, y de ejecutar código inline JavaScript. Por otro lado, el *Intérprete PsiXML* tiene la capacidad de registrar múltiples lenguajes Psi y evaluar múltiples programa Psi escritos en los diferentes lenguajes registrados.

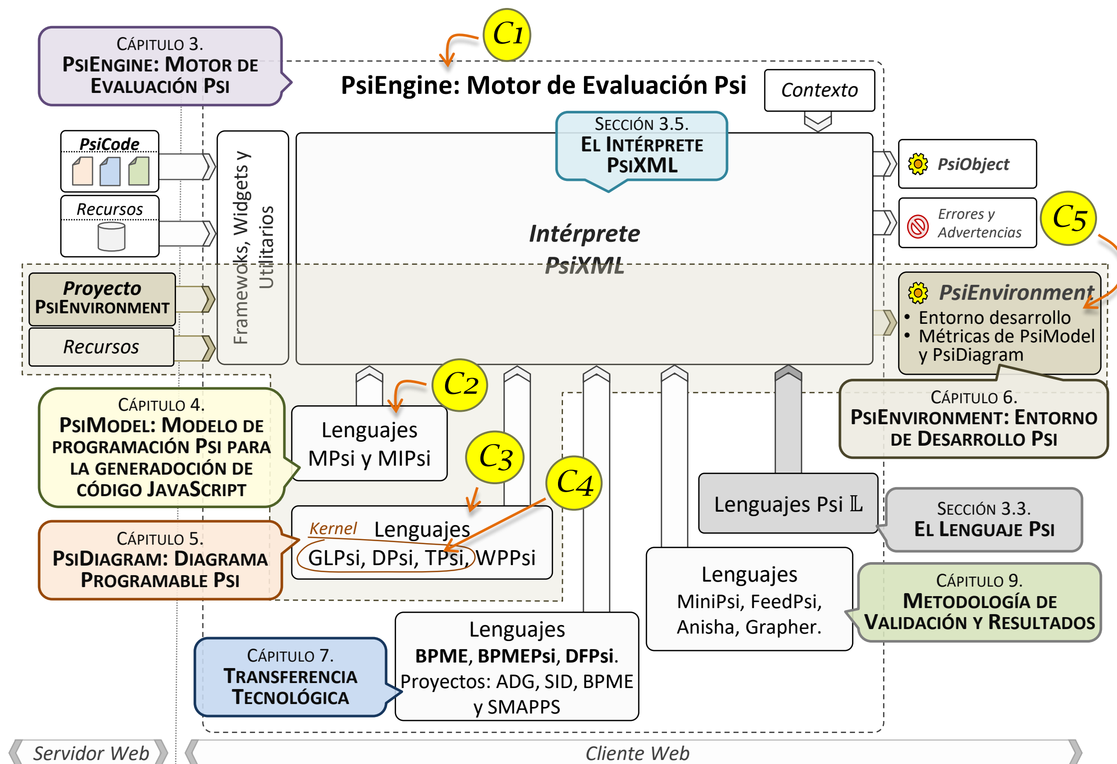


Figura 9-1. Diagrama visual de la estructura y contenido de la tesis.

Por su parte, el PsiEngine administra los programas Psi, el contexto del programa, los errores y las advertencias evaluados por el *Intérprete PsiXML*. Además, dispone de un área de trabajo para el intercambio de información y de funcionalidades entre programas Psi.

El Capítulo 3 detalla el diseño, la implementación y el funcionamiento de PsiEngine y de su *Intérprete PsiXML* (sección 3.4), además establece una metodología para el diseño y la implementación de gramáticas Psi, lenguajes Psi y componentes Psi (sección 3.3) cubriendo el primer objetivo de investigación O1.

C2. PsiModel: Modelo de programación para la creación de XML-DSLs en un CSWE

Establecido el PsiEngine, el siguiente paso consistió en facilitar la especificación e implementación de lenguajes Psi y componentes Psi. En la Ingeniería de Software, la *generación de código* es una herramienta de trabajo imprescindible, se traduce en eficiencia de programación, ahorro de tiempo, calidad, mantenibilidad y estandarización para la creación de sistemas de información.

En el Capítulo 4 se diseñó e implementó el Modelo de Programación Psi, denotado por **PsiModel**, que define un conjunto elementos de programación (variable, función, script, objeto, tipo enumerativo, clase, componente, web widget y componente Psi) creados con lenguajes Psi para la generación de código JavaScript y su documentación. Es decir, usar lenguajes Psi para crear otros lenguajes Psi. El PsiModel aplica técnicas de *code-behind* para separar la especificación (lenguaje MPsi) de su implementación (lenguaje MIPsi), para que permitan a los programadores simplificar el diseño e

implementación de funciones, objetos, componentes y componentes Psi. Este capítulo cubre el modelo de programación para la creación de XML-DSLs en un CSWE, objetivo de investigación O2.

C3. PsiDiagram: Diagrama programable para la creación de DSVLs, diagramas y generación automática de diagramas en un CSWE

La construcción de DSVLs, modelos gráficos y diagramas en un CSWE, se basa en el concepto de *diagrama programable*. Un *diagrama programable* es un conjunto de elementos gráficos que define un diagrama basado en el lenguaje SVG. Cada elemento gráfico tiene la capacidad de asociar fuentes de información heterogénea (XML y/o JSON) y unas determinadas herramientas. Estas herramientas son de dos tipos: (1) herramientas visuales, como cajas de diálogo, barras de herramientas, ventanas emergentes, soltar, arrastrar y soltar, menús y menús flotantes; y (2) elementos básicos de programación, como inclusión de ficheros y programas Psi, clases, scripts, comandos, funciones y variables.

Adicionalmente, se dispone de una herramienta para la generación de diagramas en forma automática utilizando reglas y algoritmos basados en las fuentes de información heterogéneas XML/JSON. Con esta herramienta el recomendador de diagramas genera múltiples diagramas de una fuente de información, para facilitar al usuario la selección de la visualización que sea más adecuada al problema que se desea modelar.

En el Capítulo 5 se diseñó e implementó el *diagrama programable* denominado Diagrama Programable Psi y denotado por **PsiDiagram**. El Kernel de PsiDiagram (ver Figura 9-1) está conformado por los siguientes componentes: la *Librería Gráfica Psi*, que define los elementos gráficos usando el lenguaje GLPsi; las *Herramientas Psi*, que definen las herramientas visuales, las utilidades de programación, el generador de diagramas y el recomendador de diagramas usando el lenguaje TPsi; y por último, la *Diagramación Psi*, que realiza la diagramación de los elementos gráficos sobre un lienzo SVG.

En resumen, el Capítulo 5 define y desarrolla un diagrama programable para el diseño e implementación de DSVLs, diagramas y generación automática de diagramas en un CSWE, cubriendo el objetivo de investigación O3.

C4. Mecanismos de recomendación de diagramas basados en PsiDiagram

Creado el PsiDiagram y con las herramientas para la generación automática de diagramas, se especificó el proceso para crear recomendaciones de diagramas Psi (sección 5.7). Una recomendación de diagrama en este contexto, se apoya en los diferentes estilos que se han definido para un diagrama, es decir, recomendaciones por apariencia. También se aplicaron recomendaciones por distribución del diagrama, obtenidas por los diferentes algoritmos de posicionamiento. Los mecanismos de recomendación de diagramas basados en la generación automática del diagrama programable pasaron las pruebas de concepto, cubriendo el objetivo de investigación O4.

C5. PsiEnvironment y métricas software para PsiModel y PsiDiagram

En la Ingeniería de Software una métrica representa una medida objetiva para conocer o estimar una característica de un sistema de información. A fin de poder medir las características de las aplicaciones web, componentes o widgets implementados con PsiModel y PsiDiagram, se analizaron sobre el código JavaScript resultante. El conjunto de métricas disponibles son: la complejidad ciclomática, la densidad ciclomática, las medidas de complejidad Halstead, el índice de mantenibilidad, entre otras, que denominamos *métricas básicas* (sección 6.3).

Por otro lado, en el Capítulo 6 se implementó **PsiEnvironment**, una aplicación web que establece un entorno de desarrollo ligero basado en PsiEngine para la edición de ficheros de lenguajes Psi y recursos (HTML, CSS, XML, JavaScript, Texto, SVG, entre otros). Está aplicación permite crear proyectos de tipo Genérico, PsiModel y PsiDiagram (sección 6.2.3). Un Proyecto Genérico ayuda a la administración de todo tipo de recursos. Un Proyecto PsiModel tiene como objetivo crear componentes y/o componentes Psi, basados en los lenguajes MPsi y MIPsi, y tiene la capacidad de calcular las métricas básicas adicionando las métricas propias derivadas por el modelo de programación PsiModel (sección 6.3.2). Por último, un Proyecto PsiDiagram administra la creación de diagramas, modelos gráficos y DSVLs, basados en los lenguajes GLPsi, DPsi, TPsi y WPPsi de PsiDiagram, y, tiene la capacidad de calcular las métricas básicas sobre la funcionalidad JavaScript asociada al proyecto, así como calcular un conjunto propio de métricas para PsiDiagram (sección 6.3.3).

Con estas herramientas, PsiEnvironment facilitó la implementación de PsiEngine, PsiModel y PsiDiagram y contribuyó al análisis de todos los casos de estudio para la validación de los objetivos de investigación O1, O2, O3 y O4, y en consecuencia, la validación del objetivo general de esta tesis.

9.3 GRADO DE CUMPLIMIENTO DE HIPÓTESIS

Está sección proporciona los detalles de la consecución de las hipótesis planteadas en el Capítulo 1 para nuestro trabajo de investigación. Para su validación se siguió la metodología cualitativa de casos de estudio (tipo multicaso) sugerida (Yin, 2014) y adaptada para la Ingeniería de Software en (Baxter & Jack, 2008).

En la sección 8.1 se planteó el objetivo general como el conjunto de 10 características a validar resumidas en la Tabla 8-1. Aunque fue extremadamente costoso en tiempo y ejecución, se desarrollaron un número importante de casos de estudio para validar los diferentes objetivos de investigación y las diferentes hipótesis planteadas.

H1. Es posible definir e implementar un intérprete que evalúe múltiples XML-DSLs en el CSWE

Para analizar la viabilidad de esta hipótesis, se desarrolló el PsiEngine, un motor de evaluación de programas escritos en XML-DSL que incluye el *Intérprete PsiXML*. En PsiEngine permite especificar, implementar y evaluar XML-DSLs (denominados lenguajes Psi) en un CSWE, para ello, se creó una metodología para el diseño y la implementación de gramáticas Psi, lenguajes Psi y componentes Psi. Una de las características relevantes implementadas en el *Intérprete PsiXML* es la capacidad de registrar múltiples XML-DSLs y evaluar múltiples programas de los diferentes lenguajes registrados.

La propuesta del Capítulo 3 y los casos de estudio Anisha (sección 8.2.1.3), Grapher (sección 8.2.1.4), PsiEnvironment (Capítulo 6), PsiEditDiagram (sección 8.3.1) y Proyectos de SCOOP (Capítulo 7, Proyectos ADG, SID y BPME) verifican ampliamente la validez de la hipótesis H1. Exceptuando los casos de estudio Anisha y FeedPsi que fueron usados para el artículo (R1), el resto de proyectos están actualmente en producción en B2T Concept.

H2. Es posible asociar información heterogénea XML y/o JSON de forma nativa en los XML-DSLs en el CSWE

Con el fin de abordar esta hipótesis se propusieron los lenguajes **PsiLI** (Psi Language Link Information) encargado de asociar información XML y/o JSON, y **PsiCA** (Psi Language Code Attribute) usado para evaluar código inline. Estos lenguajes se encuentran implementados en la clase *PsiElement* del Intérprete PsiXML, de la cual heredan todas las clases asociadas a cada uno de los elementos (etiquetas) de un lenguaje XML-DSL, piedra angular para la creación e implementación de cualquier Lenguaje Psi.

Todos los casos de estudio PsiModel y PsiDiagram, usan el lenguaje PsiLI para asociar información heterogénea XML y/o JSON de forma nativa a los diferentes lenguajes Psi especificados, validando constantemente la hipótesis H2. En particular, los casos de estudio MiniPsi y Anisha asocian sobre un mismo elemento o etiqueta información XML y JSON.

H3. Es posible crear un modelo de programación basado en XML-DSLs en el CSWE, para facilitar la definición de nuevos XML-DSLs

Para acometer esta hipótesis se ha propuesto el PsiModel, un modelo de programación que define un conjunto de elementos de programación basados en los lenguajes MPsi y MIPsi para generar código JavaScript. En particular, es un modelo que tiene como objetivo principal la creación de componentes Psi (sección 4.3 y 4.4), es decir la creación de nuevos XML-DSLs bajo la filosofía de PsiModel.

Adicionalmente, el PsiEnvironment tiene la capacidad de crear proyectos PsiModel, y las herramientas necesarias para la generación de componentes y componentes Psi. De los 20 casos de estudio relacionados en la Tabla 8-3, 11 proyectos generan 19 componentes Psi, es decir generan 19 XML-DSLs, verificando ampliamente la validez de la hipótesis H3.

H4. El diagrama programable permite crear modelos gráficos, generar diagramas y crear DSVLs en el CSWE

Para afrontar esta hipótesis se propone el PsiDiagram, un *diagrama programable* basado en los lenguajes GLPsi, DPsi y TPsi, que define elementos gráficos SVG, y cada elemento se le puede asociar fuentes de información XML/JSON, herramientas visuales, herramientas de programación y generador automático de diagramas.

Los proyectos PsiBase, SIDBase y BPMEBase permiten crear, editar y asociar información de 6 tipos diagramas: diagramas de clases, diagramas PsiLSD, diagramas BPME 2.0, diagramas de servicios SMAPPS, modelo de conceptos-relaciones, modelo de estados y modelo de proceso. Por otro lado, el Proyecto ADGBase permite la generación automática 8 tipos de diagramas: modelo de estados, diagramas de GANTT, diagramas de PERT, modelo de datos, árboles jerárquicos WBT y diagramas de SCOOP (llamado de funciones, mapa de objetos, modelo de conceptos-relaciones).

Como caso de estudio interesante, el proyecto SMAPPS (proyecto para la creación de aplicaciones móviles basado en servicios SMAPPS), especifica un servicio SMAPPS como un proceso de negocios usando una versión reducida y especializada de BPMN 2.0, es decir crea un DSVL basado en BPMN 2.0 para generar programas en lenguaje BPMEPsi, que posteriormente son utilizados por las aplicaciones móviles SMAPPS.

En resumen, con los proyectos mencionados anteriormente se verifica que el diagrama programable PsiDiagram crea, edita y genera diagramas, y crea DSVL, verificando rigurosamente la validez de la hipótesis H4.

H5. El uso de múltiples XML-DSLs junto con la evaluación de modelos heterogéneos dan solución a problemas de dominio específico en el CSWE

Para trabajar sobre esta hipótesis se crea PsiDiagram, que usa GLPsi, DPsi, TPsi y WPPsi para definir un diagrama programable empleando los lenguajes MPsi y MIPsi del PsiModel. A partir de ellos se crea el Proyecto SMAPPS, que define un DSVL basado en la especificación BPMN 2.0, para crear programas BPMEPsi. Estos programas creados, más un conjunto de recursos adicionales (creados por el modelo de servicios SMAPPS de SCOOP) forman parte de un *servicio SMAPPS*, que a su vez son parte de una aplicación móvil.

En conclusión, para la construcción de una aplicación móvil SMAPPS intervinieron múltiples XML-DSLs y múltiples modelos heterogéneos para su creación, verificando la validez de la hipótesis H5.

H6. Los mecanismos de recomendación para la generación de diagramas facilitan al usuario una mejor visualización para los modelos gráficos y diagramas

Para hacer frente a esta hipótesis, en PsiDiagram se implementó una herramienta para la recomendación de diagramas (sección 5.7). Esta recomendación se basa en la generación automática de diagramas de PsiDiagram y tiene dos tipos de recomendación: por estilo de diagramas o por posicionamiento de elementos gráficos. En el Proyecto ADG de SCOOP, con el diagrama de estados y el diagrama de conceptos-relaciones se establece una prueba de concepto, ya que se disponía de diferentes estilos y algoritmos de posicionamiento para verificar la validez de la H6.

- **Notas finales**

Cubiertas las hipótesis de investigación H1-H6 podemos dar por concluido el objetivo general y afirmar que se puede tener un intérprete que evalúa múltiples XML-DSLs, con asociación de información heterogénea XML y/o JSON de forma nativa en un CSWE, y que sirve para la construcción de componentes, widgets, XML-DSLs, DSVLs, modelos gráficos y diagramas que ayuden a dar soluciones a problemas específicos.

A partir del cumplimiento del objetivo general planteado en la tesis, se dispone de un motor de ejecución de programas XML-DSLs en un CSWE, de una metodología para el diseño e implementación de XML-DSLs y de un entorno de desarrollo ligero que facilitan esta labor. Además, se dispone de un diagrama programable en un CSWE para la creación de modelos gráficos, generación de diagramas y DSVLs.

Con la transferencia tecnológica realizada a través de los proyectos SCOOP, ha quedado patente la utilidad y eficacia para resolver problemas específicos, basados en la definición y construcción de XML-DSLs y DSVLs a nivel industrial. Se resuelven problemas de visualización de modelos gráficos y diagramas a partir de fuentes heterogéneas. También, se permitió establecer un modelo gráfico que realice la sincronización con información preexistente en un sistema. Por último, se resuelve la creación y edición de procesos de negocio basados en la especificación BPMN 2.0. Con un conjunto reducido de elementos gráficos BPMN se crea un DVSL que permite generar el código de una aplicación móvil híbrida basada en tecnología B2T.

Con esto se han sentado las bases para seguir construyendo soluciones basadas en la creación de XML-DSLs y diagramas programables sobre clientes web. En la siguiente sección mostraremos los trabajos futuros.

9.4 TRABAJOS FUTUROS

Internet y las nuevas tecnologías de la web 2.0, ofrecen abundantes recursos para la creación de nuevos proyectos basados en la creación de XML-DSLs y DSVLs usando como motor de evaluación PsiEngine y modelos de programación PsiModel y PsiDiagram. Los trabajos futuros pendientes por desarrollar se dividen en dos partes: (1) mantener los proyectos actuales y consolidarlos a nivel tecnológico; y (2) definir nuevas propuestas para la solución de problemas específicos.

9.4.1 CONSOLIDACIÓN DE PROYECTOS ACTUALES

La transferencia tecnológica hecha a B2T Concept implica un conjunto de tareas de mantenimiento y crecimiento de las soluciones actuales basadas en PsiEngine, PsiModel y PsiDiagram. A continuación

listaremos aquellos trabajos pendientes.

TF1. El Proyecto SID y nuevos modelos gráficos para SCOOP

Los diagramas y modelos gráficos ayudan y facilitan las labores sobre la plataforma B2T SCOOP. En la actualidad se está realizando un estudio que analice los modelos en SCOOP que son susceptibles de ser representados mediante modelos gráficos, y llegado el caso, crear nuevos DSVL.

TF2. El proyecto BPME y nuevas implementaciones basadas en procesos de negocio

Uno de los principales resultados de esta tesis fue el proyecto BPME en el que se obtuvo un diagrama programable genérico basado en BPMN 2.0. A partir de un subconjunto de elementos del BPMN, se creó un DSVL especializado para la creación de servicios SMAPPS, que a posteriori representaría una aplicación híbrida móvil. Crear un DSVL para el *BPM de Eventos* o para el *Modelo de Estados* de SCOOP basado en BPME, es otra de las alternativas que se pueden implementar.

TF3. El proyecto SMAPPS

El proyecto AngularSMAPPS requiere continuidad para la creación de nuevos elementos gráficos que amplíen el abanico posibilidades para la construcción de servicios SMAPPS y que cada vez sea más robusto. Se requiere además, que el componente AngularSMAPPSRender aumente el número de controles disponibles para la creación de páginas.

Adicionalmente, se requiere mejorar por un lado la comunicación con SCOOP para que el diagrama del servicio SMAPPS cree, edite y actualice información de SCOOP como lo soporta el Proyecto SID. Por otro lado, se requiere un lenguaje visual para el diseño, creación y renderización de páginas web, y estudiar la viabilidad de que se puedan incorporar directamente sobre el diagrama del servicio SMAPPS.

9.4.2 NUEVAS PROPUESTAS

Las nuevas tecnologías para la creación de aplicaciones web y el auge de las aplicaciones móviles permiten tener un campo amplio de acción, donde PsiEngine, PsiModel y PsiDiagram pueden sacar partido en el desarrollo de componentes, widgets, XML-DSLs y DSVLs, para solucionar problemas específicos. A continuación, algunas propuestas utilizando el enfoque propuesto:

TF4. Proyecto PsiModelVisual

El proyecto PsiBase para la creación de diagramas de clases y diagramas PsiLSD tiene los elementos necesarios para la creación de una versión visual de PsiModel. Con la experiencia obtenida en los diferentes proyectos visuales, plantear este trabajo es una tarea abordable. Con pocos recursos se puede crear un *entorno de desarrollo visual ligero* tanto para la creación de componentes JavaScript, como lenguajes Psi. Esto incrementaría el rendimiento en el diseño e implementación de lenguajes Psi.

TF5. DSLs con gramáticas JSON

Actualmente JSON es ampliamente utilizado para el intercambio de información, para la configuración y ejecución de frameworks en un entorno web. Pero ¿qué se requiere para construir un DSL con gramática JSON (JSON-DSL)? ¿Cómo asociar información heterogénea? ¿Es posible aplicar los conceptos y metodologías a los JSON-DSLs propuestos en esta tesis? Para esta propuesta surgen inquietudes interesantes.

9.5 COMENTARIOS FINALES

Los DSLs son lenguajes de programación que ofrecen abstracciones de alto nivel para facilitar el diseño, la definición y/o la implementación de sistemas de información. Cuando a un DSL lo limitamos con gramáticas XML, en ocasiones se puede perder expresividad, pero el lenguaje es fácil de usar, extensible y combinable, y puede ser usado en diferentes plataformas y sistemas de información. Además, si dotamos a los XML-DSLs con asociación de información heterogénea XML/JSON de forma nativa y lenguajes de comandos inline, los XML-DSLs son más eficientes, y producen sistemas de mayor calidad y fiabilidad. En un entorno web con las tecnologías disponibles, se pueden desarrollar componentes, widgets o aplicaciones web, como las propuestas en este trabajo.

Referencias Bibliográficas

Abílio, R., Teles, P., Costa, H. & Figueiredo, E., 2012. *A Systematic Review of Contemporary Metrics for Software Maintainability.* Natal, Brazilian, s.n., pp. 130-139.

Abiteboul, S., Segoufin, L. & Vianu, V., 2008. Static analysis of active XML systems. *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems,* pp. 221-230.

Abiteboul, S., Segoufin, L. & Vianu, V., 2009. tatic analysis of active XML systems. *ACM Transactions on Database Systems,* 34(4), pp. 1-23.

Aghaee, S. & Pautasso, C., 2014. End-User Development of Mashups with NaturalMash. *Journal of Visual Languages and Computing,* 4(25), pp. 414-432.

Agreiter, B., Hafner, M. & Breu, R., 2008. A fair Non-repudiation service in a web services peer-to-peer environment. *Computer Standards and Interfaces,* 6(30), pp. 372-378.

Ahsan, S. y otros, 2012. Object oriented metrics for prototype based languages. *Life Science Journal,* Dec., 9(4), pp. 63-66.

Alfonseca, M., De la Cruz, M., Ortega, A. & Pulido, E., 2006. *Compiladores e Intérpretes: teoría y práctica.* Madrid: Pearson Prentice Hall.

Anderson, P., 2012. *Web 2.0 and Beyond: Principles and Technologies.* London: Chapman and Hall/CRC.

Ardito, C. y otros, 2014. Creation and use of service-based Distributed Interactive Workspaces. *Journal of Visual Languages and Computing,* 6(25), pp. 717-726.

Ardito, C. y otros, 2014. User-driven visual composition of service-based interactive spaces. *Journal of Visual Languages and Computing,* 4(25), pp. 278-296.

Ariza, M. & Molina, J., 2004. *Introducción y principios básicos del desarrollo de software basado en componentes.* Bogotá: Universidad Pontificia Javeriana.

Ashkenas, J., 2010. *Backbone.JS.* [En línea] Available at: http://backbonejs.org [Último acceso: 3 2015].

Bashir, M. y otros, 2013. bugMLX: Extended Software Bug Markup Language. *11th International Conference on Frontiers of Information Technology,* pp. 217-222.

Baxter, P. & Jack, S., 2008. Qualitative Case Study Methodology: Study Design and Implementation for Novice Researchers. *The Qualitative Report*, 13(4), pp. 544-599.

Baxter, P. & Jack, S., 2008. Qualitative Case Study Methodology: Study Design and Implementation for Novice Researchers. *The Qualitative Report,* 13(4), pp. 544-599.

Bernardino, M., Zorzo, A. & Rodrigues, E., 2016. Canopus: A Domain-Specific Language for Modeling Performance Testing. *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST),* pp. 157-167.

Berrospi, E. B., 1990. *Reutilización basado en la tecnología de objetos.* s.l.:Universidad Nacional Mayor de San Marcos.

Bettini, L., 2013. *Implementing Domain-Specific Languages with Xtext and Xtend.* Birmingham: Packt Publishing Ltd.

Bettini, L., Stoll, D., Volter, M. & Colameo, S., 2013. *Approaches and Tools for Implementing Type Systems in Xtext.* Dresden, Germany, Springer Berlin Heidelberg, pp. 392-412.

Bottini, P. & Grau, A., 2004. A Suite of Metamodels as a Basis for a Classification of Visual Languages. *IEEE Symposium on Visual Languages and Human Centric Computing*, pp. 83-90.

Bottini, P., Guerra, E. & De Lara, J., 2010. language-independent and formal approach to. *IST*, 8(52), pp. 821-844.

Boyer, J., 2008. Interactive office documents: a new face for web 2.0 applications. *Proceedings of the 8th ACM Symposium on Document Engineering,* Issue 1, pp. 8-17.

Boyer, J. y otros, 2008. An office document mashup for document-centric business processes. *Proceedings of the 8th ACM Symposium on Document Engineering ,* Issue 1, pp. 100-101.

Bradley, A., 2013. *polymer.* [En línea] Available at: http://www.polymer-project.org[Último acceso: 4 2015].

Bramptom, A., 2012. *js-sequence-diagrams: Turns text into UML sequence diagrams.* [En línea] Available at: http://bramp.github.io/js-sequence-diagrams/ [Último acceso: Dic 2015].

Bravenboer, M., Trygve, K., Vermaas, R. & Visser, E., 2008. A language and toolset for program transformation. *Science of Computer Programming,* 1-2(72), pp. 52-70.

Brown, D., Levine, J. & Mason, T., 1992. *Lex & Yacc.* 2 ed. s.l.:O'Reilly Media, Inc..

Bruegge, B. & Dutoit, A., 2002. *Ingeniería de software orientado a objetos.* Primera Edición ed. México: Prentice Hall.

Bushmann, F., 1996. *Oriented Software Arquitecture. A system of patterns.* West Susses, England: ohn Wiley & Sons Ltd..

Cadavid, J., Lopez, D., Hincapié, J. & Quintero, J., 2009. A domain specific language to generate web applications. *12th Iberoamerican Conference on Requirements Engineering and Software Environments, IDEAS,* Volumen 1, pp. 1-8.

Cao, Y. & Yang, L., 2011. GISL: a generalized identity specification language based on XML schema. *Proceedings of the ACM Conference on Computer and Communications Security,* Issue 1, pp. 3-11.

Carter, Z., 2009. *JS sequence diagrams.* [En línea] Available at: http://jison.org/ [Último acceso: 2015].

Carter, Z., 2009. *Orderly.* [En línea] Available at: http://orderly-json.org/ [Último acceso: Dic 2015].

Carter, Z., 2012. *JSONLint: The JSON Validator.* [En línea] Available at: http://jsonlint.com/ [Último acceso: Dic 2015].

Carvalho, A., Oliveira, S., Fernandes, S. & Cunha, A., 2014. ASL: a DSL for remote contact center agents. *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC),* pp. 3322-3327.

Chatley, R., Ayres, J. & White, T., 2010. LiFT: Driving Development Using a Business-Readable DSL for Web Testing. *Third International Conference on Software Testing, Verification, and Validation Workshops,* pp. 460-468.

Chavarriaga, E., 2009. *Arquitectura de un sistema para la generación automática de contenidos para evaluación basados en el uso de especificaciones XML,* Madrid: s.n.

Chavarriaga, E. & Díez, F., 2008. *Sistema para la generación automática de contenido y evaluación para e-Learning, basados en el uso de herramientas XML.* Salamanca, s.n., pp. 1-7.

Chavarriaga, E. & Macías, J., 2008. *Nuevos Modelos de Presentación para la Visualización de Semántica en Interfaces de Usuario Modernas.* Albacete, España, s.n., pp. 1-10.

Chavarriaga, E. & Macías, J., 2009. A model-driven approach to building modern Semantic Web-Based User Interfaces. *Advances in Engineering Software,* 12(40), pp. 1329-1334.

Chlipala, A., 2015. From network interface to multithreaded web applications : A case study in modular program verification. *Conference Record of the Annual ACM Symposium on Principles of Programming Languages,* pp. 609-622.

Chlipala, A., 2015. From Network Interface to Multithreaded Web Applications: A Case Study in Modular Program Verification. *ACM SIGPLAN Notices,* 1(50), pp. 609-622.

Conejero, J. y otros, 2015. Automatic Configuration of Video-Surveillance Applications: a Model-Driven Experience. *IEEE Latin America Transactions,* 13(8), pp. 2700-2708.

Constantin, A., Pettifer, S. & Voronkov, A., 2013. PDFX: fully-automated PDF-to-XML conversion of scientific literature. *Proceedings of the 2013 ACM Symposium on Document Engineering ,* pp. 177-180.

Cook, S., Jones, G., Kent, S. & James, D., 2007. *Domain-Specific Development with Visual Studio DSL Tools.* s.l.:Addison-Wesley Professional.

Costagliola, G., Deufemia, V. & Polese, G., 2004. A Framework for Modeling and Implementing Visual Notations With Applications to Software Engineering. *ACM Transactions on Software Engineering and Methodology*, 13(4), pp. 431-487.

Costello, R. L., 2005. *Building Web Services the REST Way.* [En línea] Available at: http://www.xfront.com/REST-Web-Services.html [Último acceso: 5 Noviembre 2008].

Crane, D., Pascarello, E. & James, D., 2005. *Ajax in Action.* Greenwich(CT): Manning Publications.

Crockford, D., 2006. *Classical Inheritance in JavaScript.* [En línea] Available at: http://www.crockford.com/javascript/inheritance.html

Crockford, D., 2008. *JavaScript: The Good Parts.* Sebastopol(CA): Oreilly Media, Icn.

Czarnecki, K. & Eisenecker, U., 2000. *Generative Programming: Methods, Tools and Applications.* s.l.:Addison-Wesley.

de Lara, J., Guerra, E. & Cuadrado, J., 2013. Model-driven engineering with domain-specific meta-modelling languages. *Software & Systems Modeling,* 14(1), pp. 429-459.

de Lara, L. & Guerra, E., 2012. *Domain-specific textual meta-modelling languages for model driven engineering.* Lyngby, Denmark, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

Dearle, F., 2010. *Groovy for Domain-Specific Languages.* Birmingham: Packt Publishing Ltd.

Deepak, A., Crupi, J. & Dan, M., 2003. *Core J2EE Patterns.* Palo Alto, California: Sun Microsystems Press Publisher.

Delac, G. y otros, 2012. A Methodology for SIP and SOAP Integration Using Application-Specific Protocol Conversion. *ACM Transactions on the Web,* 4(6), pp. 1-15.

Demirkol, S. y otros, 2012. SEA_L: A Domain-specific Language for Semantic Web enabled Multi-agent Systems. *Federated Conference on Computer Science and Information Systems (FedCSIS),* pp. 1373-1380.

Díaz, O., Irastorza, A. C. J. & Alonso, L., 2008. From page-centric to portlet-centric Web development: Easing the transition using MDD. *Information and Software Technology,* 12(50), pp. 1210-1231.

Díez, A. y otros, 2013. *Organizations as Models, Models as Systems: the ERMM Theory of Large Scale Enterprise Executable Models.* Barcelona, s.n., pp. 19-21.

Djuric, D., Jovanovic, J., Devedzic, V. & Sendelj, R., 2010. Modeling ontologies as executable domain specific languages. *Proceedings of the 2010 India Software Engineering Conference ,* pp. 83-90.

Dojo Fundation, 2007. *dojo.* [En línea] Available at: http://dojotoolkit.org/ [Último acceso: 2 2015].

Dwijatmiko, T. & Nguyen, R., 2014. An implementation of domain specific languages to microprocessor's Memory Built in Self Repair testing. *IEEE 15th Electronics Packaging Technology Conference (EPTC 2013),* pp. 399-402.

ECMA, 2011. *ECMAScript Language Specification.* [En línea] Available at: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf [Último acceso: Febrary 2015].

Englert, B. & Lam, S., 2009. On the use of XML for port communications. *IFAC Proceedings Volumes (IFAC-PapersOnline),* pp. 50-57.

Fawcett, J., Quin, L. & Ayers, D., 2012. *Beginning XML.* 5th. ed. ed. s.l.:Wrox Press.

Figueiredo, L., Livshits, B., Molnar, D. & Veanes, M., 2016. Prepose: Privacy, Security, and Reliability for Gesture-Based Programming. *2016 IEEE Symposium on Security and Privacy (SP),* pp. 122-137.

Firmenich, S., Rossi, G., Winckler, M. & Palanque, P., 2014. An approach for supporting distributed user interface orchestration over the Web. *International Journal of Human Computer Studies,* 1(72), pp. 53-76.

Fowler, M., 2010. *Domain Specific Languages.* s.l.:Addison-Wesley.

Fowler, M. & White, T., 2010. *Domain Specific Languages.* s.l.:Addison-Wesley Professional.

Fredrick, C., 2003. Extreme Programming: Growing a Team Horizontally . En: *Extreme Programming and Agile Methods - XP/Agile Universe 2003.* s.l.:Springer: Lecture Notes in Computer Science, pp. 9-17.

Galin, D., 2004. *Software Quality Assurance: From theory to implementation.* Edinburgh Gate, England: Pearson Education Limited.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J., 2003. *Patrones de Diseño.* Reimpresión 2006 ed. Madrid: Addison Wesley.

García, F., Marqués, J. & Maudes, J., 1997. *Análisis y diseño orientado al objeto para reutilización.* Burgos: Universidad de Burgos.

Ghosh, D., 2010. *DSLs in Action.* Greenwich(CT): Manning Publications.

Gill, G. & Kemerer, C., 1991. Cyclomatic Complexity Density and Software Maintenance Productivity. *IEEE Transactions on Software Engineering,* Dec., 17(12), pp. 1284-1288.

Goodliffe, P., 2006. *Code Craft.* San Francisco: No Starch Press, Inc.

Goodman, D., Morrison, M., Novitski, P. & Gustaff, T., 2010. *JavaScript® Bible.* 7th ed. ed. Washington: John Wiley & Sons.

Google, 2010. *AngularJS.* [En línea] Available at: https://angularjs.org/ [Último acceso: Mar. 2015].

Granback, R., 2009. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.* s.l.:Addison-Wesley Professional.

Greenfield, J. & Short, K., 2004. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools.* s.l.:Wiley Publishing.

Groenewegen, D., Hemel, Z. & Visser, E., 2010. Separation of concerns and linguistic integration in webDSL. *IEEE Software,* 5(27), pp. 31-37.

Groenewegen, D. M. & Visser, E., 2009. Weaving web applications with WebDSL: (demonstration). *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications (OOPSLA '09)*, pp. 797-798.

Groenewegen, D. & Visser, E., 2008. Declarative Access Control for WebDSL: Combining Language Integration and Separation of Concerns. *8th International Conference on Web Engineering, ICWE 2008 ,* Issue 1, pp. 175-188.

Groenewegen, D. & Visser, E., 2013. Integration of data validation and user interface concerns in a DSL for web applications. *Software and Systems Modeling,* 1(12), pp. 35-52.

Guerra, E., 2007. *Especificación, análisis y generación de entornos para Lenguajes Visuales de Dominio Específico.* Madrid: Universidad Autónoma de Madrid.

Guerra, E. y otros, 2013. Engineering model transformations with transML. *Software and Systems Modeling,* Jul, 12(3), pp. 555-577.

Halstead, M., 1977. *Elements of Software Science.* New York: The Computer Science Library.

Havelund, K., 2014. Data Automata in Scala. *2014 Theoretical Aspects of Software Engineering Conference,* pp. 1-9.

Heer, F. & Agrawala, M., 2006. Software Design Patterns for Information Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5).

Hermida, J., Meliá, S. & Arias, A., 2016. Xanui: A textual platform-independent model for rich user interfaces. *Journal of Web Engineering,* 1-2(15), pp. 45-83.

Herzberg, D. y otros, 2009. Specifying computer-based counseling systems in health care: A new approach to user-interface and interaction design. *Journal of Biomedical Informatics,* 2(42), pp. 347-355.

He, W., LvT., Meis, M. & Yan, P., 2013. Visual Evaluation of XPath Queries. *International Conference on Computational and Information Sciences,* pp. 434-437.

He, W., Lv, T., Meis, M. & Yan, P., 2012. VXPath: A Visual XPath Query Evaluator. *Fourth International Conference on Computational and Information Sciences,* pp. 361-364.

Hlaoui, Y. & Benayed, L., 2008. A MDA approach for semi automatic grid services workflows composition. *IEEE International Conference on Industrial Engineering and Engineering Management, IEEM 2008 ,* pp. 1433-1437.

Holdener III, A., 2007. What Are Mashups?. En: *Ajax: The Definitive Guide.* Primera Edición ed. s.l.:O'Reilly Media, Inc..

Holzner, S., 2006. *Ajax For Dummies®.* s.l.:John Wiley & Sons.

Hudak, P., 1996. Building domain-specific embedded languages. *ACM Comput. Surv.,* 4(196), pp. 28-42.

Hu, H. & Yan, G., 2010. Research on DSL-based composition language in service-oriented architecture. *International Conference on Computer Application and System Modeling (ICCASM 2010),* Volumen 14, pp. 357-360.

IMS QTI, 2008. *IMS Question & Test Interoperability Specification.* [En línea] Available at: http://www.imsglobal.org/question/ [Último acceso: 16 Noviembre 2008].

Jacob, F., Wynne, A., Liu, Y. & Gray, J., 2014. Domain-Specific Languages for Developing and Deploying Signature Discovery Workflows. *Computing in Science and Engineering,* 1(16), pp. 52-64.

Jatain, A. & Mehta, Y., 2014. *Metrics and models for Software Reliability: A systematic review.* Ghaziabad, India, s.n., pp. 210-214.

Jingjing, L. & Chunlin, P., 2012. *jQuery-based Ajax general interactive architecture.* Chengdu: Software Engineering and Service Science (ICSESS).

Jouault, F., Allilaire, F., Bézivin, J. & Kurtev, I., 2008. ATL: A model transformation tool. *Science of Computer Programming,* Jun, 72(1-2), pp. 31-39.

Karam, M. y otros, 2008. A product-line architecture for web service-based visual composition of web applications. *Journal of Systems and Software,* 6(81), pp. 855-867.

Karlsson, E.-A., 1995. *Software reuse. A Holistic Approach.* s.l.:John Wiley & Sons Ltd..

Kats, L., Kalleberg, K. & Visser, E., 2009. *Domain-Specific Languages for Composable Editor Plugins.* s.l., Elsevier, pp. 149-163.

Kelker, R., 2013. *Clojure for Domain-specific Languages.* Birmingham: Packt Publishing Ltd.

Kelly, S. & Tolvanen, J., 2008. Domain-Specific Modeling: Enabling Full Code Generation. *Wiley-IEEE Computer Society Press*.

Kern, C., 2014. Securing the tangled web. *Commun. ACM*, Sep, 57(9), pp. 38-47.

Kleppe, A., 2008. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels.* 1st ed. ed. Boston: Addison-Wesley Professional.

Klint, P., Storm, T. & Vinju, J., 2013. *RASCAL: A Domain Specific Language for Source Code Analysis and Manipulatio.* Eindhoven, Países Bajos, IEEE, pp. 168-177.

Kniesel, G., Winter, V., Siy, H. & Zand, M., 2009. Making aspect-orientation accessible through syntax-based language composition. *In IET Software. IEEE,* 3(1), pp. 1-13.

Kolweyh, M. & Lechner, U., 2006. Towards P2P Information Systems. En: *Lecture Notes in Computer Science: Innovative Internet Community Systems.* Berlin / Heidelberg: Springer, pp. 79-90.

Kosar, T., Bohra, S. & Mernik, M., 2016. Domain-Specific Languages: A Systematic Mapping Study. *Information and Software Technology,* Issue 71, pp. 77-91.

Kosar, T. y otros, 2010. Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems,* 7(2), p. 247–264.

Kourie, D., Fick, D. & Watson, B., 2008. Virtual machine framework for constructing domain-specific languages. *In IET Software. IEEE,* 3(3), pp. 219-237.

Lachgar, M. & Abdali, A., 2015. Generating Android graphical user interfaces using an MDA approach. *Colloquium in Information Science and Technology, CIST ,* pp. 80-85.

Lee, P., 2014. *CoffeeScript in Action.* Greenwich, USA: Manning Publications Co..

Levine, J., 2009. *Flex & Bison.* s.l.:O'Reilly Media, Inc..

Liu, Q., Bryant, B. & Mernik, M., 2010. Metamodel Recovery from Multi-tiered Domains Using Extended MARS. *IEEE 34th Annual Computer Software and Applications Conference,* Volumen 288, p. 279.

Liu, S. y otros, 2010. A SOA Approach for Domain-Specific Language Implementation. *Proceedings - 2010 6th World Congress on Services, Services-1 2010 ,* pp. 535-542.

Liu, Y., Liu, Z. & Luo, L., 2009. Heterogeneous data integration framework based on grid service. *Proceedings of 2009 IEEE International Conference on Network Infrastructure and Digital Content, IEEE IC-NIDC2009 ,* pp. 871-874.

Maeda, K., 2009. Ruby-based data representation and the performance in Java programs. *2nd International Conference on the Applications of Digital Information and Web Technologies, ICADIWT ,* pp. 814-819.

Mahmoudi, C. & Mourlin, F., 2014. Business Process Management with mobile routes. *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA ,* pp. 420-427.

Mahmoud, T., Rapp, B. & Van Vliet, S., 2014. Web Service Integration within Next Generation CEMIS. *International Journal of Web Portals,* 2(6), pp. 59-74.

Marín de la Iglesia, J., 2010. *Web 2.0: Una descripción muy sencilla de los cambios que estamos viviendo.* 1º ed. La Coruña: Netbiblo S.L..

Matsumoto, Y., 2008. *Ruby: A Programmer's Best Friend.* [En línea] Available at: http://www.ruby-lang.org/es/ [Último acceso: 5 Noviembre 2008].

Maximilien, E., Ranabahu, A. & Gomadam, K., 2008. An Online Platform for Web APIs and Service Mashups. *IEEE Internet Computing,* 5(12), pp. 32-43.

McCabe, T., 1976. A Complexity Measure. *IEEE Transactions on Software Engineering,* Dec., SE-2(4), pp. 308-320.

McDaniel, A., 2011. Your visual blueprint™ for designing rich web pages and applications. En: Indianapolis(NJ): Jhon Wiley & Sons, Inc..

McGuire, P., 2007. *Getting Started with Pyparsing.* s.l.:O'Reilly Media Inc..

Mellor, S., Clark, A. & Futagami, T., 2003. Guest Editors' Introduction: Model-Driven Development. Volumen 20, pp. 14-18.

Mernik, M., Heering, J. & Sloane, A., 2005. When and how to develop domain-specific languages. *ACM Comput. Surv.,* 4(37), p. 316–344.

Mernik, M., Lenic, M., Avdičaušević, E. & Zumer, V., 2002. LISA: An Interactive Environment for Programming Language Development. *Springer Berlin Heidelberg, Lecture Notes in Computer Science,* 23(4), pp. 1-4.

Microsoft, 2014. *Microsoft ASP.NET.* [En línea] Available at: http://www.asp.net/Microsoft, 2015. *Modeling SDK.* [En línea] Available at: https://www.microsoft.com/en-us/download/details.aspx?id=48148 [Último acceso: Enero 2016].

Miles, R. & Hamilton, K., 2006. *Learning UML 2.0.* s.l.:O'Reilly Media, Inc..

Mishra, J. & Mohanty, A., 2011. *Web Modeling Language.* s.l.:Pearson Education India.

Mistrik, I. y otros, 2015. *Software Quality Assurance.* Waltham, MA: Lindsay Lawrence.

Montesi, F., L. I. & Mazzara, M., 2015. The evolution of Jolie: From orchestrations to adaptable choreographies. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),* pp. 506-521.

Montilva, J., Arapé, N. & Colmenares, J., 2003. *Desarrollo de software basado en componentes.* Merida, s.n.

Nguyen, V., Deeds-Rubin, S., Tan, T. & Boehm, B., 2007. *A SLOC Counting Standard,* s.l.: s.n.

Nicolette, D., 2015. *Software Development Metrics.* Shelter Island, NY: Manning Publications Co..

Nolan, B. y otros, 2008. *Model Driven Systems Development with Rational Products.* New York: IBM Redbooks.

Northover, S. & Wilson, M., 2004. *SWT: The Standard Widget Toolkit.* s.l.:Addison Wesley Professional.

Oliveira, A., Araújo, J. & Amaral, V., 2010. The VisualAORE DSL. *Fifth International Workshop on Requirements Engineering Visualization,* pp. 11-19.

Oman, P. W., Hagemeister, J. & and Ash, D., 1991. *A Definition and Taxonomy for Software Maintainability,* Moscow: s.n.

OMG, 2009. *Documents Associated With UML Version 2.2.* [En línea] Available at: http://www.omg.org/spec/UML/2.2/ [Último acceso: Febrero 2013].

OMG, 2011. *Business Process Model and Notation (BPMN) Versión 2.0.* [En línea] Available at: http://www.omg.org/spec/BPMN/2.0/PDF/ [Último acceso: Feb 2015].

OMG, 2011. *Documents Associated With Business Process Model And Notation (BPMN) Version 2.0.* [En línea] Available at: http://www.omg.org/spec/BPMN/2.0/

OMG, 2016. *OMG: MDA Specifications.* [En línea] Available at: http://www.omg.org/mda/specs.htm [Último acceso: Nov. 2016].

OPML, 2007. *Outline Processor Markup Language (OPML) 2.0.* [En línea] Available at: http://www.opml.org/spec2 [Último acceso: 5 Noviembre 2008].

Oracle, 2003. *JavaServer Pages Technology.* [En línea] Available at: http://www.oracle.com/technetwork/java/javaee/jsp/index.html [Último acceso: 2015].

Oracle, 2014. *Oracle JavaServer Face.* [En línea] Available at: javaserverfaces.java.net

O'Reilly, T., 2005. *What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software.* [En línea] Available at: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html [Último acceso: 5 Noviembre 2008].

Oren, E., Heitmann, B. & Decker, S., 2008. ActiveRDF: Embedding Semantic Web data into object-oriented languages. *Web Semantics,* 3(2), pp. 191-202.

Osmani, A., 2012. *Smashing Magazine: Journey Through The JavaScript Jungle.* [En línea] Available at: http://www.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle-2/

Osmani, A. y otros, 2012. *TodoMVC.* [En línea] Available at: http://todomvc.com/ [Último acceso: Feb. 2015].

Oyarzun, D. y otros, 2009. ADML: A framework for representing inhabitants in 3D virtual worlds. *Proceedings of Web3D 2009: The 14th International Conference on Web3D Technology ,* pp. 83-90.

Pant, K. & Juric, M., 2008. *Business Process Driven SOA using BPMN and BPEL: From Business Process Modeling to Orchestration and Service Oriented Architecture.* Birmingham: Packt Publishing.

Parr, T., 2013. *The Definitive ANTLR 4 Reference.* 2nd ed. ed. Raleigh(NC): Pragmatic Bookshelf.

Pender, T., 2003. *UML™ Bible.* s.l.:John Wiley & Sons.

Pérez, M., Sanz, I. & Berlanga, R., 2010. XTaGe: a flexible XML collection generator. *Proceedings of the ACM SIGMOD International Conference on Management of Data,* pp. 1139-1142.

Placek, P. y otros, 2008. A heuristic approach for checking containment of generalized tree-pattern queries. *International Conference on Information and Knowledge Management, Proceedings,* pp. 551-560.

Prud, E., Gayo, J. & Solbrig, H., 2014. Shape expressions: an RDF validation and transformation language. *ACM International Conference Proceeding Series,* pp. 32-40.

Rahien, A., 2010. *DSLs in Boo: Domain-Specific Languages in .NET.* Greenwich(CT): Manning Publications Co.

Ramirez, A., 2004. *Introducción a los patrones de diseño.* Stanford: Creative Commons.

Reid, J. & Valentine, T., 2013. *JavaScript Programmer's Reference.* 1 ed. New York: Apress Media LLC.

Riaz, M., Mendes, E. & Tempero, E., 2009. *A systematic review of software maintainability prediction and metrics.* Lake Buena Vista, FL, s.n., pp. 367-377.

Ribes, X., 2007. *La Web 2.0. El valor de los metadatos y de la inteligencia colectiva.* [En línea] Available at: https://telos.fundaciontelefonica.com/telos/articuloperspectiva.asp@idarticulo=2&rev=73.htm [Último acceso: 21 Noviembre 2008].

Richards, J., Riley, O., Ekstrom, J. & Tew, K., 2013. Termediator: early studies in terminological mediation between disciplines. *RIIT 2013 - Proceedings of the 2nd Annual Conference on Research in Information Technology ,* pp. 1-9.

RSS Specifications, 2007. *RSS Specifications.* [En línea] Available at: http://www.rss-specifications.com/rss-specifications.htm [Último acceso: 5 Noviembre 2008].

Ruiz, J., 2010. *Compiladores: teoría e implementación.* Madrid(Madrid): RC Libros.

Sarasa, A. & Sierra, J.-L., 2015. A syntax-directed model transformation framework based on attribute grammars. *Communications in Computer and Information Science,* pp. 145-152.

Satu, S., Parvez, H. & Al-Mamun, S., 2015. Review of integrated applications with AIML based chatbot. *2015 International Conference on Computer and Information Engineering (ICCIE),* pp. 87-90.

Schmidt, D., 2006. Model-Driven Engineering. *IEEE Computer,* 39(2).

Serrano, A., 2014. *Beginning Haskell A Project-Based Approach.* s.l.:Apress.

Shim, H., Kang, B. & Kwag, K., 2009. Web2Animation - Automatic Generation of 3D Animation from the Web Text. *Proceedings - 2009 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2009 ,* Volumen 1, pp. 596-601.

Shu, J. y otros, 2008. Unification of problem solving environment implementation layers with XML-based specifications. *Advances in Engineering Software,* 3(39), pp. 189-201.

Sommaruga, L., Formilli, T. & Rizzo, N., 2011. DomoML: an integrating devices framework for ambient intelligence solutions. *ACM International Conference Proceeding Series ,* pp. 9-15.

Spinellis, D., 2001. Notable design patterns for domain-specific languages. *Journal of Systems and Software,* 56(1), pp. 91-99.

Stefanov, S. & Chetan, K., 2013. *Object-Oriented JavaScript.* Birmingham: Packt Publishing.

Steinberg, D., Budinsky, F., Paternostro, M. & Merks, E., 2008. *EMF: Eclipse Modeling Framework.* s.l.:Addison-Wesley Professional.

Stilwell, J., 2015. *npm: Escomplex.* [En línea] Available at: https://www.npmjs.com/package/escomplex [Último acceso: 2016].

SVG: Basic Shapes, 2011. *W3C Recomendation.* [En línea] Available at: http://www.w3.org/TR/SVG/shapes.html [Último acceso: Agosto 2014].

SVG: Painting, 2011. *W3C.* [En línea] Available at: http://www.w3.org/TR/SVG/painting.html [Último acceso: 2014].

SVG: Paths, 2011. *W3C Recomendation.* [En línea] Available at: http://www.w3.org/TR/SVG/paths.html [Último acceso: Febrero 2014].

SVG: Text, 2011. *W3C Recommendation.* [En línea] Available at: http://www.w3.org/TR/SVG/text.html [Último acceso: Agosto 2014].

Taddesse, F. & Chbeir, R., 2010. Semantic aware RSS query algebra. *International Conference on Information Integration and Web-Based Applications and Services ,* pp. 291-298.

Tahir, A. & MacDonell, S. G., 2012. *A systematic mapping study on dynamic metrics and software quality.* Trento, s.n., pp. 326-335.

Tamayo, A., Granell, C. & Huerta, J., 2011. Instance-based XML data binding for mobile devices. *Proceedings of the 3rd International Workshop on Middleware for Pervasive Mobile and Embedded*

Computing, M-MPAC 2011 - Co-located with ACM/IFIP/USENIX International Middleware Conference , pp. 1-8.

Tarazona, G., Pelayo, B., Sanjuán, O. & Rodríguez, L., 2014. e-Commerce Metamodel Based on MDA. *IEEE Latin America Transactions,* 4(12), pp. 699-705.

Teorey, T., Lightstone, S., Nadeau, T. & Jagadish, H., 2011. *Database Modeling and Design.* s.l.:Morgan Kaufmann.

Testing, L. D. D. U. a. B.-R. D. f. W., 2010. Chatley, R.; Ayres, J.; White, T.. *Third International Conference on Software Testing, Verification, and Validation Workshops,* pp. 460-468.

Thies, G. & Vossen, G., 2009. Modelling web-oriented architectures. *Conferences in Research and Practice in Information Technology Series ,* Volumen 96, pp. 1-7.

Thomo, A. & Venkatesh, S., 2008. Rewriting of visibly pushdown languages for XML data integration. *International Conference on Information and Knowledge Management, Proceedings,* pp. 521-530.

Thu Thuy, P., Lee, Y. & Lee, S., 2009. DTD2OWL: automatic transforming XML documents into OWL ontology. *ACM International Conference Proceeding Series ,* Issue 403, pp. 125-131.

Tsalaportas, P., Kapinas, V. & Karagiannidis, G., 2015. Solar Lab Notebook (SLN): An Ultra-Portable Web-Based System for Heliophysics and High-Security Labs. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing,* 8(8), pp. 4141-4150.

Tulsian, P. & Pandey, V., 2006. *Quantitative Techniques: Theory and Problems.* Tamil Nadu: Pearson India.

Van Den Brand, M., Heering, J., Klint, P. & Olivier, P., 2002. Compiling language definitions: the ASF+SDF compiler. *ACM Transaction Programming Languages and Systems,* July, 24(4), pp. 334-368.

Van Duyne, D. K., Landay, J. & Hong, J. I., 2006. Permalinks. En: *The Design of Sites: Patterns for Creating Winning Web Sites.* Second Edition ed. s.l.:Prentice Hall.

Visser, E., 2008. WebDSL: A Case Study in Domain-Specific Language Engineering. *Generative and Transformational Techniques in Software Engineering II: Lecture Notes in Computer Science*, pp. 291-373.

Vlist, E. y otros, 2007. *Professional Web 2.0 Programming.* s.l.:Wiley Publishing.

Voelter, M. y otros, 2013. *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages.* s.l.:Dslbook.org.

Vrieze, P. y otros, 2011. Building enterprise mashups. *Future Generation Computer Systems,* 5(27), pp. 637-642.

W3C: HTML & CSS, 2010. *HTML5 (Hypertext Markup Language) and CSS (Cascading Style Sheets).* [En línea] Available at: http://www.w3.org/standards/webdesign/htmlcss [Último acceso: Frebrero 2012].

W3C: JavaScript Web APIs, 2010. *DOM (Document Object Model), XBL (XML Binding Language), WAI-ARIA (Acessible Rich Internet Applications) and Mobile Web Application.* [En línea] Available at: http://www.w3.org/standards/webdesign/script

W3C: MathML, 2003. *Mathematical Markup Language (MathML) Version 2.0 (Second Edition).* [En línea] Available at: http://www.w3.org/TR/2003/REC-MathML2-20031021/ [Último acceso: 4 Noviembre 2008].

W3C: SMIL 3.0, 2008. *Synchronized Multimedia Integration Language (SMIL 3.0).* [En línea] Available at: http://www.w3.org/TR/SMIL3/ [Último acceso: 5 Noviembre 2008].

W3C: Transformation, 2010. *XSL (eXtensible Stylesheet Language), XSLT (XSL Transformations) y XSL-FO (XSL Formating Object).* [En línea] Available at: http://www.w3.org/standards/xml/transformation [Último acceso: Enero 2012].

W3C: XHTML, 2000. *XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition).* [En línea] Available at: http://www.w3.org/TR/xhtml1/ [Último acceso: 4 Noviembre 2008].

W3C: XLink, 2001. *XML Linking Language (XLink) Version 1.0.* [En línea] Available at: http://www.w3.org/TR/xlink/ [Último acceso: 5 Noviembre 2008].

W3C: XML Technology, 2010. *XML Essentials, Schema, Security, Transformation, Query, Components, Processingm, Internationalization and Publishing.* [En línea] Available at: http://www.w3.org/standards/xml/ [Último acceso: Febrero 2012].

W3C: XML, 2008. *Extensible Markup Language (XML) 1.0 (Fifth Edition).* [En línea] Available at: http://www.w3.org/TR/xml/ [Último acceso: 2012].

W3C: XPath, 2007. *XML Path Language (XPath) 2.0.* [En línea] Available at: http://www.w3.org/TR/xpath20/ [Último acceso: 5 Noviembre 2008].

W3C: XQuery, 2007. *XQuery 1.0 and XPath 2.0 Data Model (XDM).* [En línea] Available at: http://www.w3.org/TR/xpath-datamodel/ [Último acceso: 5 Noviembre 2008].

W3C: XSL, 2007. *XSL Transformations (XSLT) Version 2.0.* [En línea] Available at: http://www.w3.org/TR/xslt20/ [Último acceso: 4 Noviembre 2008].

W3C, 2004. *Document Object Model (DOM).* [En línea] Available at: http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/ [Último acceso: Feb 2015].

W3C, 2011. *Scalable Vector Graphics (SVG) 1.1 Specification.* [En línea] Available at: http://www.w3.org/TR/SVG11/ [Último acceso: March 2015].

W3C, 2014. *HTML & CSS, Scripting and Ajax, Graphics, Audio and Video, Accessibility, Internationalization, Mobile Web, Privacy and Math on the Web.* [En línea] Available at: http://www.w3.org/standards/webdesign/ [Último acceso: Febrary 2015].

Ward, M., 1995. Language-oriented programming. *Software Concepts Tools,* Issue 15, p. 147–161.

White, A., 2009. *JavaScript Programmer's Reference.* Indianapolis(IN): Wiley Publishing, Inc..

Williams, I., 2009. *Beginning XSLT and XPath: Transforming XML Documents and Data.* Indianapolis(IN): WileyPublishing, Inc..

Winblad, A., Edwards, S. & King, D., 1990. *Software orientado a objetos.* s.l.:Addison-Wesley Iberoamericana y Ediciones Diaz de Santos, S.A..

Wu, L. & Li, Y., 2012. Investigations on XML-based Data Exchange between Heterogeneous Databases. *Ninth Web Information Systems and Applications Conference,* pp. 21-24.

Yahya, F., Boukadi, K., Maamar, Z. & Abdallah, H., 2015. Enhancing business processes with Web 2.0 features. *2015 12th International Joint Conference on e-Business and Telecommunications (ICETE),* Volumen 2, pp. 183-190.

Yin, R., 2014. *Case Study Research: Design and Methods.* 5ª ed. London, United Kingdom: Sage Publications.

Yue, C. & Wang, H., 2013. A measurement study of insecure javascript practices on the web. *ACM Transaction Web*, May, 7(2), pp. 1-29.

Zampoglou, M. y otros, 2013. Direct mapping of X3D scenes to MPEG-7 descriptions. *Proceedings - Web3D 2013: 18th International Conference on 3D Web Technology ,* pp. 57-66.

Zhang, G. & Hölzl, M., 2015. Improving the modularity of web-application models with aspects. *International Journal of Web Engineering and Technology,* 2(10), pp. 98-128.

Anexo A. Generación de Código Psi

A.1. Fuentes de generación Psi del tipo enumerativo JavaScript

En la Figura A-1 se muestran los fuentes de generación Psi para el tipo enumerativo, su etiqueta es **Enum** y los atributos básicos se definen en la Tabla 4-1. La Implementación MIPsi contiene la lista de miembros (etiqueta **Member**) del tipo enumerativo. Cada miembro define un nombre (atributo *name*) y un valor (atributo *value*). Por ejemplo, si es un valor es numérico entonces `value="5"` o si es una cadena de caracteres `value="'yes'"`. Nótese que en la plantilla *Enum* durante el proceso de generación, {{members}} de la plantilla *Enum* debe ser sustituido por el resultado de la concatenación de la plantilla *Member* tantos miembros existan en la etiqueta **Enum**. El campo {{separator}} es coma "," mientras no sea el último miembro de la lista.

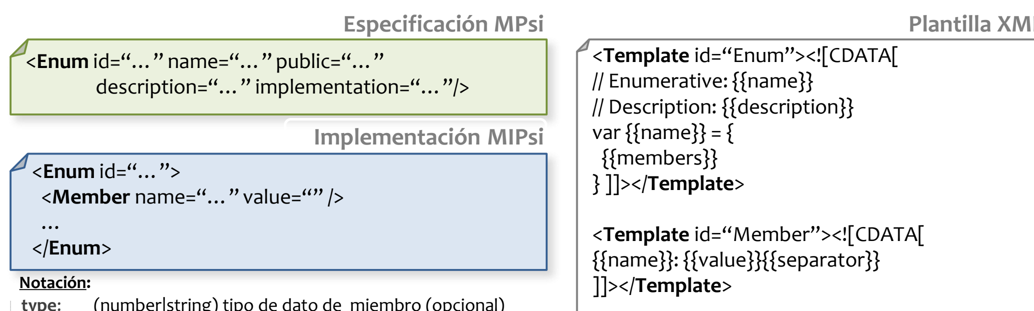


Figura A-1. Fuentes de generación Psi para un tipo enumerativo de JavaScript.

A.2. Fuentes de generación Psi de un objeto JavaScript

En la Figura A-2 se muestran los fuentes de generación Psi para un objeto de JavaScript, su etiqueta es **Object** y los atributos básicos se definen en la Tabla 4-1. La Especificación MPsi de un objeto está constituida por las etiquetas:

- Etiqueta **Properties**: lista de propiedades del objeto. Cada atributo representa una propiedad del objeto, donde el nombre de atributo es el nombre de la propiedad y su valor es la descripción.
- Etiqueta **Methods**: lista de métodos del objeto. Al igual que las propiedades, cada atributo representa el nombre de un método con su descripción.

Por otro lado, la Implementación MIPsi de un objeto está constituida por las etiquetas:

- Etiqueta **Property**: implementación de cada propiedad definida en la especificación. Cada propiedad se identifica por su nombre (atributo *name*) y su valor por defecto (atributo *default*).
- Etiqueta **Method**: implementación de cada método del objeto. Cada método tiene definido su nombre (atributo *name*, debe existir en la definición), sus argumentos (atributo *arguments*) y su cuerpo (contenido del elemento).

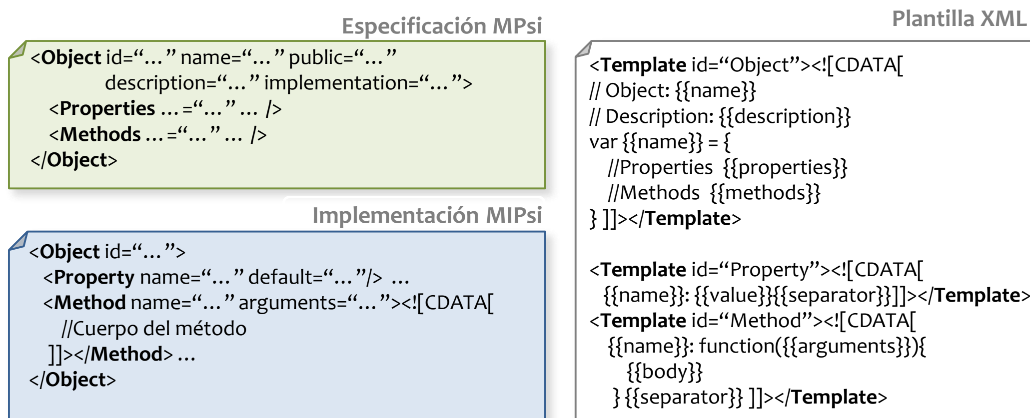


Figura A-2. Fuentes de generación Psi para un objeto de JavaScript.

Se definen tres plantillas para la generación del código de un objeto JavaScript:

- *Object*: contiene el esqueleto del objeto.
- *Property*: contiene el esqueleto de una propiedad.
- *Method*: contiene el esqueleto de un método.

En su generación se aplica la plantilla *Property*, tantas propiedades existan, se concatenan y se cambian en {{properties}} de la plantilla *Object*. Del mismo modo se generan los métodos sobre {{methods}} de la plantilla *Object*.

A.3. Fuentes de generación Psi de un Web Widget

En la Figura A-3 se muestran las fuentes de generación Psi para un web widget de jQuery UI, su etiqueta es **Widget**. Además de los atributos básicos definidos en la Tabla 4-1, se define el widget padre jQuery UI (atributo *inherits*) y la exclusión del Widget de la generación (atribute *exclude*). La Especificación MPsi de un web widget está constituida por:

- Etiqueta **Options**: lista opciones con sus descripciones para el web widget jQuery. Las opciones son los elementos de configuración del widget.
- Etiqueta **Properties**: lista de propiedades con sus descripciones de la clase.
- Etiqueta **Events**: lista de eventos a sobrescribir del widget jQuery.
- Etiqueta **Methods**: lista de métodos con sus descripciones del widget jQuery.

La Implementación MIPsi del web widget está constituida por las etiquetas:

- Etiqueta **Option**: implementación de cada opción definida en la especificación. Cada opción se identifica por su nombre (atributo *name*) y su valor por defecto (contenido de la etiqueta).
- Etiqueta **Property**: implementación de cada propiedad definida en la especificación. Cada propiedad se identifica por su nombre (atributo *name*) y su valor por defecto (atributo *default*).

- Etiqueta **Event**: implementación de cada evento a sobrescribir del widget jQuery. El evento tiene definidos sus argumentos (atributo *arguments*) y su cuerpo (contenido de la etiqueta).
- Etiqueta **Method**: implementación de método de la clase. El método tiene definido sus argumentos (atributo *arguments*) y su cuerpo (contenido de la etiqueta).

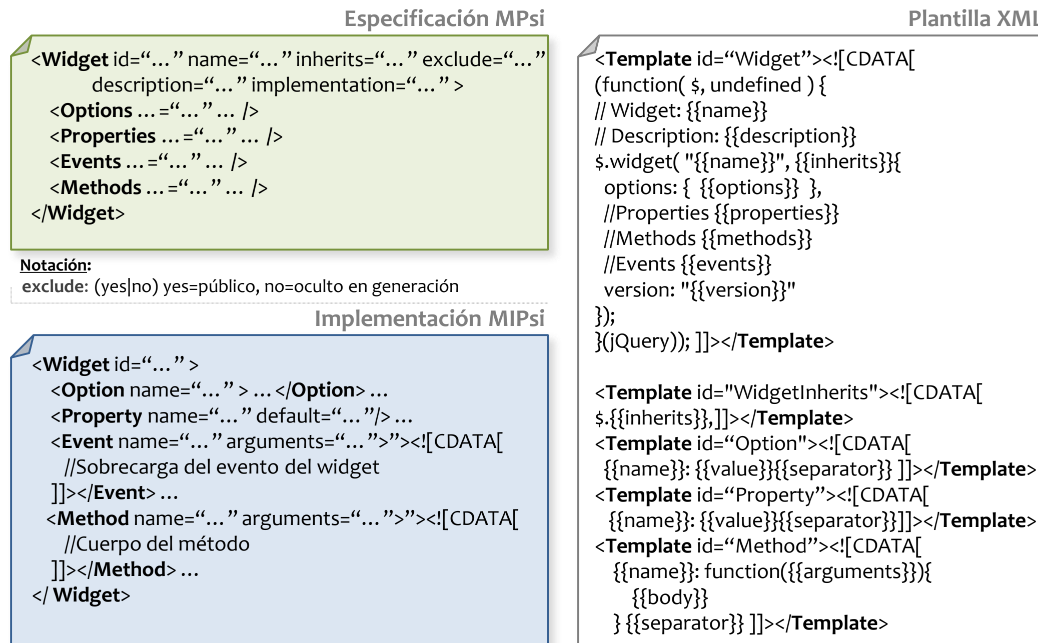


Figura A-3. Fuentes de generación Psi para un widget jQuery.

Se definen cinco plantillas para generar una web widget jQuery UI:

- *Widget*: contiene el esqueleto del web widget.
- *WidgetInherits*: contiene el esqueleto para definir la superclase widget jQuery UI.
- *Option*: contiene el esqueleto de una opción.
- *Property*: contiene el esqueleto de una propiedad.
- *Method*: contiene el esqueleto de un método o un evento.

Para la generación de una web widget, se aplican las diversas plantillas según sea el caso: se aplica la plantilla *WidgetInherits*, si el atributo *inherits* está definido y se reemplaza en {{inherits}} de la plantilla *Widget*. Y se aplican las plantillas *Option*, *Property*, *Event* y *Method*, tantas veces existan, se concatenan y se cambian respectivamente en {{options}}, {{properties}}, {{events}} y {{methods}} de la plantilla *Widget*.

Anexo B. Especificaciones Psi

B.1. Especificación GLPsi/DPsi de una Figura Psi

En la Figura B-1 se muestra la plantilla de Especificación GLPsi y la plantilla de Especificación DPsi para la **Figura Psi**.

- **Especificación GLPsi**

El GLPsi posee la etiqueta **Defs** para describir los *símbolos* y la *forma* de la Figura. Está etiqueta fue descrita en la especificación de Línea Psi (Sección 5.5.1.1).

El *borde de conexión* se define con la etiqueta **ConnectionEdge** en GLPsi como se muestra en la Figura B-1, y puede ser usada tanto en Figura, como en Lista y Contenedor. La etiqueta **ConnectionEdge** tiene un identificador (atributo *id*) y una clase de estilos CSS para su *forma* (atributo *class*). Contiene una lista *bordes* y cada *borde* define un conjunto de puntos del *borde de conexión*. Un *borde* se especificada con la etiqueta **Edge** y puede ser de dos tipos:

i. Conjunto de puntos (atributo *type*=points).
ii. Segmento de recta (atributo *type*=segment).

Cada *borde* tiene un identificador (atributo *id*) y la fuente de cómo construir el *borde* (atributo *source*). Para construir el *borde* se tiene como referencia el rectángulo delimitador[12] de un elemento SVG seleccionado (atributo *selector*) de la Figura (Lista o Contenedor) o el mismo elemento agrupador **g** de SVG de la Figura (Lista o Contenedor). Para acceder al rectángulo delimitador, el *borde* genera las variables: X, Y (origen), W (ancho) y H (alto). Por ejemplo, en la Figura 5-9 tomemos el rectángulo (elemento **rect**[key=box]) como la referencia al *borde* de la Figura, al definir *source*="W/2,0;W/2,H;0,H/2;W,H/2", significa que los puntos del *borde* son los puntos medios de los lados del rectángulo.

Para visualizar el *borde de conexión* de tipo puntos, se usan círculos SVG (elemento **circle**) cuyo centro son los puntos generados por la fuente, su radio (atributo *r*) y su estilo (atributo *class*) son tomados de la etiqueta **Edge**. Por otro lado, para visualizar un *borde* de tipo segmento de recta (elemento **line** de SVG), la fuente define los puntos extremos mientras que con el atributo *class*, el estilo.

Una **Figura** es definida mediante la etiqueta **Shape**, tiene un identificador (atributo *id*), un punto medio (atributo *midpoint*), la referencia a un *borde de conexión* (atributo *connection*) y una clase de

[12] Para todo elemento SVG se puede obtener su rectángulo delimitador (interface *SVGRect*) conformado por el punto de origen (atributos *x* e *y*), su ancho (atributo *width*) y su alto (atributo *height*). Este rectángulo se obtiene con el método *getBBox* del elemento, ver (SVG: Types and Interfaces 2011).

estilos CSS (atributo *class*). Contiene los *elementos básicos* (etiqueta **SourceSVG**), un *ícono* (etiqueta **IconSVG**), la *herramienta de escala y rotación* (etiqueta **Settings**) y los *eventos* (etiqueta **Events**).

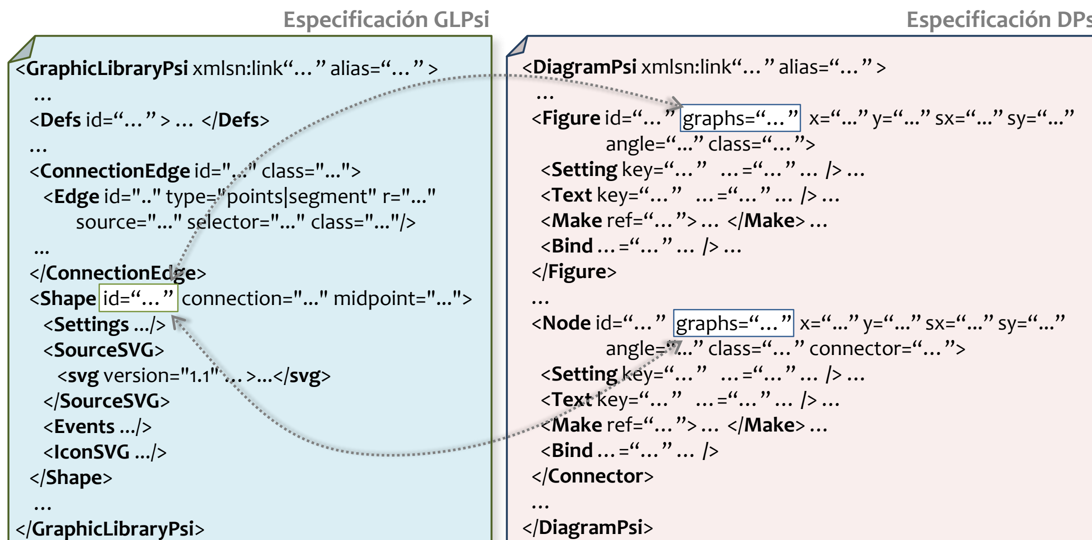


Figura B-1. Especificación de Figura Psi en el lenguaje GLPsi y en el lenguaje DPsi.

Los *elementos básicos* son agrupados en un elemento **g** de SVG, y su especificación se encuentra en la etiqueta **SourceSVG**, la cual define un documento SVG con los elementos básicos SVG que conforman la Figura (ver Figura B-1). A cada elemento SVG se asocia, si es necesario, el atributo *key* como clave única de identificación dentro de la Figura, con el objetivo de referenciarlos para las herramientas de modificación, etiquetas **Setting** y/o **Make** del DPsi.

En términos generales, la etiqueta **IconSVG** define con elementos SVG el *ícono* de un elemento gráfico Psi. Son útiles para la creación de menús, la notación en pie de diagramas, etc. La etiqueta **Settings** configura la *herramienta de escala y rotación* del elemento gráfico Psi. Los atributos *scale-min* y *scale-max* determinan el tamaño mínimo y máximo del elemento. El atributo *scale-class* da estilo CSS al rectángulo de la herramienta y el atributo *scale-r* el radio de los círculos SVG para generar la escala en forma dinámica. El atributo *box-class* define el estilo CSS del rectángulo de selección.

Los *eventos* para manipular un elemento gráfico Psi son especificados mediante la etiqueta **Events**. Se definen cuatro tipos de evento: evento de ocultar la Figura (atributo *hidden*), evento de selección de Figura (atributo *select*), evento para ocultar/mostrar el *borde de conexión* (atributo *toggle-connection*) y el evento para visualizar la herramienta de rotación y redimensionamiento de la Figura (atributo *transform*). Estos eventos son aplicables a Lista y Contenedor.

- **Especificación DPsi**

El DPsi contiene dos etiquetas para pintar una Figura especificada en GLPsi, la etiqueta **Figure** y la etiqueta **Node**. Su diferencia radica en que a la etiqueta **Figure** no se le define un *borde de conexión,* y es usada para el pintado de objetos o elementos decorativos o aclarativos de la Figura (por ejemplo títulos, márgenes, comentarios, etc.). Mientras que con la etiqueta **Node** se establece un vínculo con otros elementos gráficos a través de conectores.

Las etiquetas **Figure** y **Node** contienen las etiquetas: **Setting**, **Text**, **Make** y **Bind**. La etiqueta **Setting** tiene la capacidad de modificar los elementos SVG de la Figura a través de su clave (atributo *key*). Por ejemplo, modificar el color, grosor, relleno, texto, etc., de los *elementos básicos*.

La etiqueta **Text** se especializa en el texto de una línea o multilínea. Mientras que la etiqueta **Make** es una función especializada para agregar, quitar o modificar *elementos básicos* de la Figura. También se usa para agregar funcionalidades de movimiento, de decoración, de llamado a funciones, etc. Por último, la etiqueta **Bind** es la que asocia herramientas y utilidades Psi.

B.2. Especificación GLPsi/DPsi de una Lista Psi

En la Figura B-2 se muestra un ejemplo de implementación de la definición de Lista Simple en SVG. Se usa como agrupador el elemento **g** de SVG con un identificador (atributo *id*). Los *elementos básicos*, los *símbolos* y *forma* son los mismos definidos que en Figura. Para el *marco* de la Lista se usa el elemento **rect** de SVG y su apariencia se define con estilo CSS. En caso de redimensionamiento de la Lista, el *marco* se ajusta al nuevo tamaño. Por otro lado, los *ítems* son agrupados por el elemento **g** de SVG indicando su origen después del encabezado (atributo *transform*). Cada *ítem* tiene un agrupador **g** de SVG donde se define el origen (atributo *transform*) con respecto al agrupador de *ítems*, y contiene sus *elementos básicos*. Cuando se agrega *secciones*, cada *sección* al igual que el *ítem*, tiene un agrupador **g** de SVG con su origen (atributo *transform*) y sus *elementos básicos*.

Representación de Lista en SVG

```
<svg width="640" height="480" xmlns="http://www.w3.org/2000/svg"
   xmlns:xlink="http://www.w3.org/1999/xlink">
 <defs>
                                                                    Símbolos
  <style type="text/css">
   svg {font-family: Arial, Tahoma, Geneva, sans-serif; font-size: 0.9em;}
   .background  {fill:#ffffff; stroke:gray; stroke-width:0.4px;}
   .title {font-weight: bold; font-size: 0.9em; text-anchor:middle;}
   .box {fill: #fcfce0; stroke: #cccccc; stroke-width: 0.5px;}
   .circle {stroke: gray; stroke-width: 0.5px;}
  </style>                                                            Forma
 </defs>
 <g id="icon" class="icon">
A <rect key="background" x="0" y="0" width="96" height="65" rx="4" ry="4"
        class="background"/>/
B <g key="header" transform="translate(3,3)">
   <rect key="header-box" class="box" x="0" y="0" width="90" height="20" rx="3"/> #
   <image key="settings" x="2" y="4" width="12" height="12" xlink:href="settings.png"/> #
   <text key="header-title" class="title" x="48" y="14">Lista</text> #
  </g>
N <g key="items" transform="translate(3,26)">
   <g key="item1" transform="translate(0,2)">
I   <text key="item-label" x="4" y="12" class="class-item-label">Item 1</text> @
    <circle key="item-color" cx="80" cy="7" r="5" fill="yellow" class="circle"/> @
   </g>
   <g key="item2" transform="translate(0,19)">
I   <text key="item-label" x="4" y="12" class="class-item-label">Item 2</text> @
    <circle key="item-color" cx="80" cy="7" r="5" fill="blue" class="circle"/> @
   </g>
  </g>
 </g>                                                                   Lista
</svg>
```

Resultado en SVG

Lista
Item 1
Item 2

A *Marco de Lista*
B *Encabezado de Lista*
Elementos Básicos de Encabezado
N *Ítems de Lista*
I *Ítem de Lista*
@ *Elementos Básicos de Ítem*

Figura B-2. Ejemplo de implementación de la representación de Lista en SVG.

En la Figura B-3 se muestra la plantilla de Especificación GLPsi y la plantilla de Especificación DPsi para la **Lista Psi**.

■ Especificación GLPsi

El GLPsi posee la etiqueta **Defs** y la etiqueta **ConnectionEdge** para describir los *símbolos*, la *forma* y el *borde de conexión*. Estas etiquetas fueron descritas en la especificación de Línea Psi y Figura Psi.

Una **Lista** se define mediante la etiqueta **List**, tiene un identificador (atributo *id*), la referencia al *borde de conexión* (atributo *connection*), el origen en el diagrama (atributos *x* e *y*), el ancho de la lista (atributo *width*) y una clase de estilos CSS (atributo *class*). El alto de una Lista es calculado automáticamente según la suma de los altos de los ítems y secciones.

La Lista contiene los *elementos básicos* de: el *encabezado* (etiqueta **HeaderSVG**), la plantilla de *sección* (etiqueta **SectionSVG**, opcional) y la plantilla de *ítem* (etiqueta **ItemSVG**). Al igual que la etiqueta **SourceSVG** de Figura, cada una de ellas contiene un documento SVG con sus respectivos *elementos básicos* SVG, a los cuales se le pueden definir el atributo *key* para referenciarlos en las herramientas de modificación.

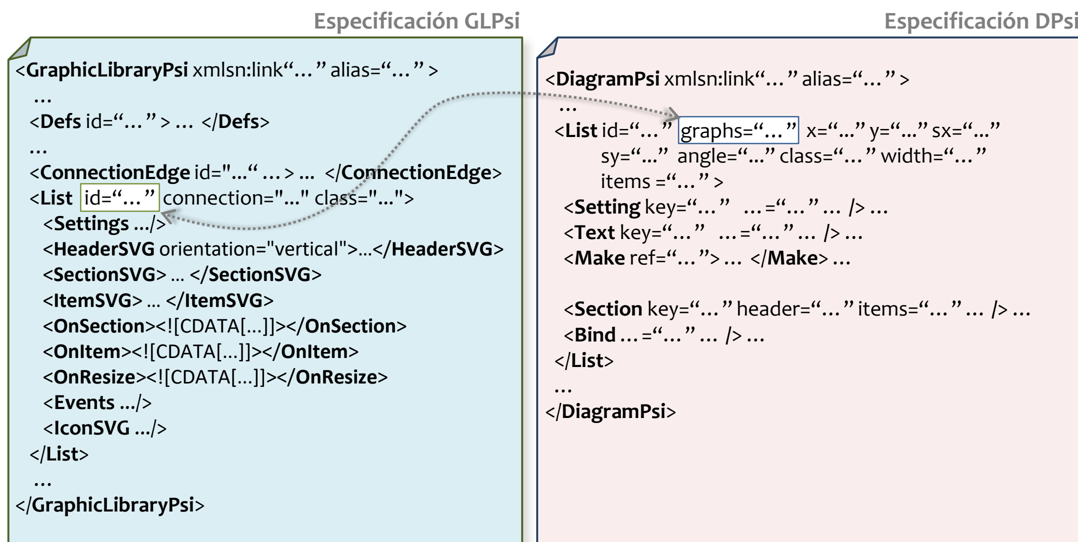


Figura B-3. Especificación de Lista Psi en el lenguaje GLPsi y en el lenguaje DPsi.

La Lista contiene además un *ícono* (etiqueta **IconSVG**), la *herramienta de escala y rotación* (etiqueta **Settings**) y los *eventos* (etiqueta **Events**), que fueron explicadas en la definición de Figura Psi.

Las etiquetas **OnSection** y **OnItem** se usan cuando una *sección* o un *ítem* son creados, respectivamente. Es código JavaScript que tiene la finalidad de personalizar la *sección* o el *ítem*, por ejemplo, alinear, modificar textos y formas de los elementos básicos. La etiqueta **OnResize** es usada para manipular *ítems* y *secciones* cuando hay un cambio de tamaño en la Lista.

Por último, los *eventos* para manipular un elemento gráfico Psi son especificados en la etiqueta **Events**, y son los mismos explicados en la definición de Figura.

■ Especificación DPsi

El DPsi contiene la etiqueta **List** para pintar una Lista especificada en GLPsi. Tiene un identificador único dentro del diagrama (atributo *id*) y una referencia a la definición de Lista en GLPsi (atributo *graph*), como se muestra en la Figura B-3. Contiene las etiquetas: **Setting**, **Text**, **Make**, **Bind** y **Section**. Las primeras cuatro etiquetas fueron descritas en la definición de Figura. La etiqueta **Section** tiene una clave de identificación (atributo *key*), el título de la *sección* (atributo *header*, opcional) y un selector de *ítems* (atributo *items*) para determinar cuales pertenecen a la *sección*.

La construcción de una Lista en un diagrama SVG es sencilla, primero se crea la *cabecera* y con su altura se establece el origen del agrupador de *ítems*. Se obtiene la lista de *ítems* y para cada *ítem* se clonan los *elementos básicos* de **ItemSVG** de GLPsi, se modifica con la información respectiva del *ítem* (Uso de **OnItem** de GLPsi) y se calcula su altura. La posición de un *ítem* se desplaza en vertical con la suma de las alturas de los anteriores *ítems*. Por último, se redimensiona el *marco* y se aplica **OnResize** de GLPsi. En caso de que existan *secciones*, para cada *sección* se clona los *elementos básicos* de **SectionSVG** y se ubican entre medias de los *ítems*, según corresponda.

B.3. Especificación GLPsi/DPsi de un Contenedor Psi

En la Figura B-4 se muestra un ejemplo de implementación de la definición de Contenedor en SVG. Nuevamente, se usa como agrupador el elemento **g** de SVG con un identificador (atributo *id*). Los *elementos básicos*, los *símbolos* y *forma* son los mismos definidos que en Figura y Lista.

Para el *marco* del Contenedor se usa el elemento **rect** de SVG y su apariencia se define con CSS, además es redimensionable al cambio de tamaño. Para la *cabecera* y el *pie* se usa el agrupador **g** de SVG

Definición de Contenedor en SVG

```
<svg width="640" height="480" xmlns="http://www.w3.org/2000/svg">
 <defs>
                                                         Símbolos
  <style type="text/css">
   svg {font-family: Arial, Tahoma, Geneva, sans-serif; font-size: 0.9em;}
   .background {fill:#ffffff; stroke:gray; stroke-width:0.4px;}
   .title {font-weight: bold; font-size: 0.8em; text-anchor:middle;}
   .box {fill: #f2fce8; stroke: #bfceaf; stroke-width: 0.5px;}
   .border {stroke: gray; stroke-width: 0.5px;}
   .footer {font-size: 0.6em;font-weight: bold;}                Forma
  </style>
 </defs>
 <g id="psi_texts" transform="translate(50 50) scale(1 1) rotate(0)">
(A)<rect x="0" y="0" width="279" height="250" rx="4" ry="4" class="background"/>
  <g key="header" transform="translate(3,3)">
(H) <rect key="header-box" class="box" x="0" y="0" width="273" height="20" rx="4"/>(1)
    <text key="header-title" class="title" x="115" y="15">Container</text(1)
  </g>
(L)<g id="left" transform="translate(3,26)">
    <svg id="svg-left" x="0" y="0" width="50" height="210">
(M)  <rect key="left-box" x="0" y="0" width="50" height="250" fill="#f0f0f0"/>
(3)  <g id="app-1" transform="translate(0,0)">
       <rect x="15" y="12" width="15" height="15" rx="4" ry="4" fill="yellow" class="border"/>
       <text key="item-label" x="22.5" y="35" class="title">App 1</text>
     </g>
    </svg>
  </g>
(B)<g id="psi_texts-gbody" transform="translate(56,26)">
    <svg id="svg-body" x="0" y="0" width="180" height="210" viewBox="0 0 175 210">
(3)  <g id="node-1" transform="translate(30,40)">
       <circle cx="15" cy="12" r="10" fill="blue" class="border"/>
     </g>
(3)  <g id="node-2" transform="translate(100,40)">
       <circle cx="15" cy="12" r="10" fill="green" class="border"/>
     </g>
(3)  <g id="line-id"><path key="path" d="m55,52l50,0" class="border"/></g>
    </svg>
  </g>
(R)<g id="right" transform="translate(236,26)">
    <svg id="svg-right" x="0" y="0" width="40" height="210">
(M)  <rect key="left-box" x="0" y="0" width="50" height="250" fill="#E3F0F9"/>
    </svg>
  </g>
  <g key="footer" transform="translate(3,232)">
(F) <text key="header-title" x="4" y="15" class="footer">My Footer!!</text>(2)
  </g>
 </g>                                                     Contenedor
</svg>
```

Resultado en SVG

(A) *Marco de Contenedor*
(H) *Cabecera de Contenedor*
(1) *Elementos de Cabecera*
(F) *Pie de Contenedor*
(2) *Elementos de Pie*
(L) *Lateral Izquierdo*
(R) *Lateral Derecho*
(M) *Marco Lateral*
(B) *Cuerpo del Contenedor*
(3) *Elementos Gráficos* (Líneas, Figuras, Listas, etc.)

Figura B-4. Ejemplo de implementación de la representación de Contenedor en SVG.

y cada uno contiene sus *elementos básicos*.

Los *laterales* y el *cuerpo* del Contenedor están definidos en un agrupador con el elemento **g** de SVG. Cada uno en su interior tiene un elemento **svg** de SVG que es el contenedor de elementos geométricos SVG, tiene un identificador (atributo *id*), la posición relativa al origen del contenedor (atributos *x* e *y*), el ancho (atributo *width*), el alto (atributo *height*) y el marco de vista SVG (atributo *viewBox*). Tanto en los *laterales* como en el *cuerpo* se definen Líneas, Figuras, Listas y Contenedores.

Como ejemplo, en la Figura B-4 se muestra, en el *lateral izquierdo*, una Figura (identificador "app-1"), en el *cuerpo* del Contenedor dos Figuras (identificador "node-1" y "node-2") y una Línea (identificador "line-1"), por último, en el *lateral derecho* no hay elementos gráficos.

En la Figura B-5 se muestra la plantilla de Especificación GLPsi y la plantilla de Especificación DPsi para el **Contenedor Psi**.

- **Especificación GLPsi**

El GLPsi dispone de la etiqueta **Defs** y la etiqueta **ConnectionEdge** para describir los *símbolos*, la *forma*, y el *borde de conexión*. Estas etiquetas fueron descritas en la especificación de Figura Psi y Lista Psi.

Un **Contenedor** es definido mediante la etiqueta **Container**, tiene un identificador (atributo *id*), la referencia a un *borde de conexión* (atributo *connection*) y una clase de estilos CSS (atributo *class*). Este Contenedor agrupa los *elementos básicos* de: el *encabezado* (etiqueta **HeaderSVG**) y el *pie de contenedor* (etiqueta **FooterSVG**, opcional). Al igual que la etiqueta **SourceSVG** de Figura, cada una de ellas contiene un documento SVG con sus respectivos elementos básicos SVG, a los cuales se les puede incluir el atributo *key* para referenciarlos en las herramientas de modificación.

El Contenedor admite además, un *ícono* (etiqueta **IconSVG**), la *herramienta de escala y rotación* (etiqueta **Settings**) y los *eventos* (etiqueta **Events**). Todas ellas ya fueron explicadas en la definición de Figura Psi. Los *eventos* son los mismos descritos en la definición de Figura.

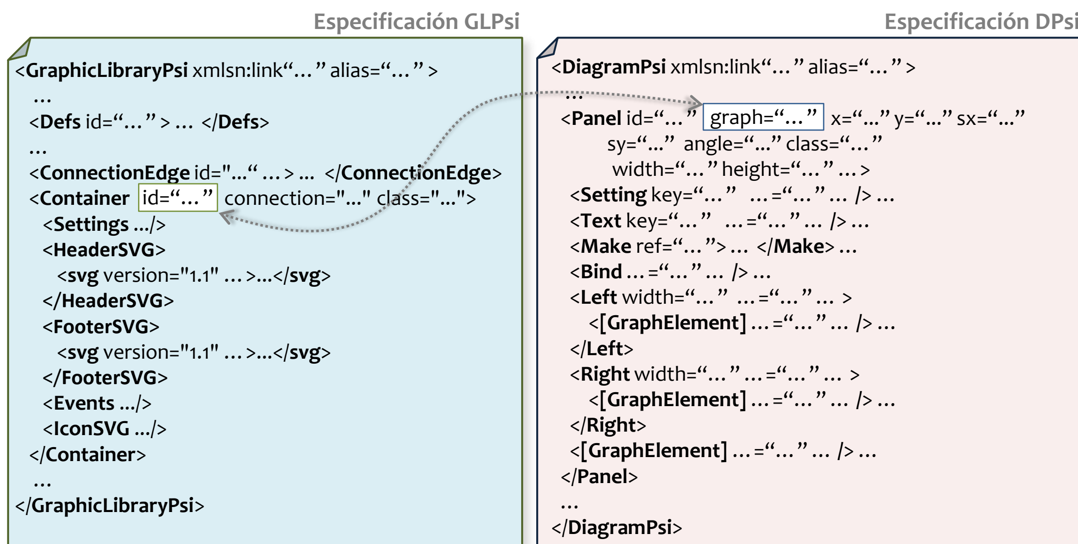


Figura B-5. Especificación de Contenedor Psi en el lenguaje GLPsi y en el lenguaje DPsi.

- **Especificación DPsi**

El DPsi contiene la etiqueta **Panel** para pintar un Contenedor especificado en GLPsi. Tiene un identificador único dentro del diagrama (atributo *id*), el origen en el diagrama (atributos *x* e *y*), el ancho

del *cuerpo* (atributo *width*), el alto del *cuerpo* (atributo *height*) y una referencia a la definición de Contenedor en GLPsi (atributo *graph*), como se muestra en la Figura B-5.

La etiqueta **Panel** contiene las etiquetas: **Setting**, **Text**, **Make**, **Bind**, **Left**, **Right** y **[GraphElement]** (elementos gráficos de diagrama **Figure**, **Node**, **Line**, **Connector**, **List** y **Panel**). Las primeras cuatro etiquetas fueron descritas en la definición de Figura. Las etiquetas **Left** y **Right** tienen un ancho (atributo *width*) y elementos gráficos de diagrama. El ancho del Panel se calcula con la suma de sus *laterales* y su *cuerpo*, mientras que el alto se calcula con suma del alto del *encabezado*, el alto *pie de contenedor* y el alto del *cuerpo*. El *marco* se redimensiona con el ancho y alto del Panel. El *borde de conexión* esta delimitado por el *marco*.

B.4. Especificación GLPsi de las etiquetas IconSVG, Settings y Events

En el FRAGMENTO B-1 se muestran las plantillas para las etiquetas **IconSVG**, **Settings** y **Events** del Lenguaje GLPsi. La denotación de etiqueta [**ElementGraph**] representa el conjunto de todos los elementos gráficos Psi: *Línea Psi*, *Figura Psi*, *Lista Psi* y *Contenedor Psi*.

La etiqueta **IconSVG** define con elementos SVG el ícono de un elemento gráfico Psi. Son útiles para reconocer el elemento, por ejemplo, para la creación de menús, la notación en pie de diagramas, etc. En la definición del ícono se aconseja el uso de símbolos SVG.

La etiqueta **Settings** configura la herramienta de escala y rotación del elemento gráfico Psi. Los atributos *scale-min* y *scale-max* determinan el tamaño mínimo y máximo del elemento. El atribute *scale-class* da estilo CSS al rectángulo de la herramienta y el atributo *scale-r* el radio de los círculos SVG para generar la escala en forma dinámica.

La etiqueta **Events** define los siguientes eventos básicos:

- **Evento de movimiento** (atributo *drag*="[selector]"), con [selector] se determina(an) el/los elemento(s) encargado(s) para realizar el movimiento del elemento gráfico Psi.
- **Evento de selección**, (atributo *select*="[selector]|[event]"), con [selector] se selecciona el/los elemento(s) encargado(s) para seleccionar el elemento gráfico Psi, consiste en remarcar con un rectángulo el elemento. Esta acción de ejecuta mediate un evento [event] de clic (valor “click”) o doble-clic (valor “dblclick”) sobre los elementos seleccionados.
- **Evento de ocultación**, (atributo *select*="[selector]|[event]"), con [selector] se selecciona el/los elemento(s) encargado(s) para ocultar el elemento gráfico Psi. Esta acción se define mediate un evento [event] de clic (valor “click”) o doble-clic (valor “dblclick”) sobre los elementos seleccionados.

FRAGMENTO B-1. Plantilla para las etiquetas **Settings**, **IconSVG** y **Events** del Lenguaje GLPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<GraphsLibrary xmlns:xlink="http://www.w3.org/1999/xlink" alias="psi">
  ...
  <[ElementGraph]>
    <Settings width="…" background-class="…" background-padding="…" background-rx="…"
              background-ry="…" space-header="…" space-item="…" divider-class="…" .../>
    <IconSVG>
      <svg version="1.1" xmlns="http://www.w3.org/2000/svg" width="…" height="…" ...>...</svg>
    </IconSVG>
    <Events drag="[selector]" select="[selector]|[event]" hide="[selector]|[event]"/> ...
  </[ElementGraph]>
  ...
</GraphsLibrary>
```

El FRAGMENTO B-2 es una extensión de la plantilla anterior para los elementos gráficos Psi a los cuales se les pueden definir bordes de conexión: *Figura Psi*, *Lista Psi* y *Contenedor Psi*. La denotación de [**ElementGraphWithEdge**] representa estos elementos. La extensión contiene la plantilla para las etiquetas **ConnectionEdge**, **Edge**, **Events** (atributos adicionales) y **Commands**.

La etiqueta **ConnectionEdge** define un borde de conexión con un identificador (atributo *id*) y una clase de estilos CSS (atributo *class*). Un borde de conexión esta constituido por un conjunto de bordes (etiquetas **Edge**). Un borde puede ser de dos tipos: conjunto de puntos (atributo *type*=points) y segmento de recta (*type*=segment). Cada borde tiene un identificador (atributo *id*) y la fuente de cómo construir el borde (atributo *source*) teniendo como referencia un elemento SVG seleccionado (atributo *selector*). Para visualizar el borde de tipo puntos se usan círculos SVG (elemento **circle**), el centro de cada punto lo genera la fuente, mientras que el radio y el estilo son los atributos *r* y *class*, respectivamente. Por otro lado, para visualizar un borde de tipo segmento de recta (elemento **line**), la fuente define los puntos extremos mientras que con el atributo *class*, da el estilo.

Los elementos gráficos Psi [**ElementGraphWithEdge**] definen un borde de conexión haciendo referencia al identificador de la etiqueta **ConnectionEdge** (atributo *connection*). En la etiqueta **Events** se pueden adicionar los siguientes eventos:

- **Evento de ocultar/mostrar el borde de conexión**, (atributo *select*="[selector]|[event]"), con [selector] se selecciona el/los elemento(s) encargado(s) que mostrará el borde de conexión mediante un evento [event] de clic (valor “click”) o doble-clic (valor “dblclick”) de los elementos seleccionados.
- De forma análoga, el evento para visualizar la herramientas de redimensionamiento y rotación del el elemento gráfico Psi se define con el atributo *transform*="[selector]|[event]".

La etiqueta **Commands** será explicada en la diagramación de elementos gráficos Psi con los lenguajes DPsi y TPsi.

FRAGMENTO B-2. Plantilla para las etiqueta ConnectionEdge, Edge, Events y Commands del Lenguaje GLPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<GraphsLibrary xmlns:xlink="http://www.w3.org/1999/xlink" alias="psi">
  ...
  <ConnectionEdge id="…" class="…">
    <Edge id="…" type="points|segment" source="…" r="…" selector="…" class="…"/> ...
  </ConnectionEdge> ...
  <[ElementGraphWithEdge] connection="…">
    <Events toggle-connection="[selector]|[event]" transform="[selector]|[event]"/>
    <Commands [name]="[selector]|[event]:[command]" ...=""/> ...
  </[ElementGraphWithEdge]>
  ...
</GraphsLibrary>
```

B.5. Especificación GLPsi de las etiquetas Function, Script, Make y Defs

En el FRAGMENTO B-3 se muestra la plantilla para las etiquetas **Function**, **Script** y **Make** del Lenguaje GLPsi. Por un lado, la etiqueta **Script** se usa para definir el cuerpo de un script en lenguaje JavaScript. La etiqueta **Function** encapsula y resuelve una tarea específica en lenguaje JavaScript, usualmente para el apoyo de diagramación de elementos gráficos. Tiene el identificador (atributo *id*) y la lista de argumentos separadas por coma “,” (atributo *arguments*). Por último, la etiqueta **Make** es el modificador de elementos gráficos. Esta etiqueta tiene un identificador (atributo *id*) que unido con el alias de GLPsi forma un nombre y puede ser utilizado por uno varios elemento(s) gráfico(s) en el momento de

diagramación. Al modificador se le pasa tanto los elementos SVG, como datos asociados para crear un procedimiento de personalización del elemento gráfico Psi.

FRAGMENTO B-3. Plantilla para las etiquetas Funcion, Script y Make del Lenguaje GLPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<GraphsLibrary xmlns:xlink="http://www.w3.org/1999/xlink" alias="psi">
  ...
  <Script id="…"><![CDATA[
    //Cuerpo del script
  ]]></Script> ...
  <Function id="…" arguments="…"><![CDATA[
    //variables: self, context
  ]]></Function> ...
  <Make id="…"><![CDATA[
    //variables: parent, data
  ]]></Make> ...
  ...
</GraphsLibrary>
```

En el FRAGMENTO B-4 se muestra la plantilla para la etiqueta **Defs** del Lenguaje GLPsi. Tiene un identificador (atributo *id*) y un conjunto de etiquetas **SourceSVG**. Para cada etiqueta **SourceSVG** se adiciona un documento SVG con elementos **defs** de SVG, para crear marcadores (elemento **marker**), líneas de gradiente (elemento **linearGradient**), filtros (elemento **filter**), símbolos (elemento **simbol**) y estilos CSS (elemento **style**). Estas definiciones son usadas en el momento de diagramación para los elementos gráficos Psi, por ejemplo, en los extremos de una línea se usan marcadores para dar el aspecto de flechas, conectores, etc.

FRAGMENTO B-4. Plantilla para la etiqueta Defs del Lenguaje GLPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<GraphsLibrary xmlns:xlink="http://www.w3.org/1999/xlink" alias="psi">
  <Defs>
    <SourceSVG id="…">
      <svg version="1.1" xmlns="http://www.w3.org/2000/svg">
        <defs id="…">
          <marker id="…" ...>...</marker> ...
          <linearGradient id="…" ...>...</linearGradient>...
          <filter id="…" ...="…" ...>...</filter>...
          <simbol id="…" ...="…">...</simbol>
          <style type="text/css"><![CDATA[/*Cuerpo CSS*/]]></style>...
        </defs> ...
      </svg>
    </SourceSVG> ...
  </Defs>
  ...
</GraphsLibrary>
```

B.6. Especificación DPsi de los Utilidades de definición

En el FRAGMENTO B-5 se muestra la plantilla para las etiquetas **Params**, **Bind**, **Include**, **Make** y **SettingsN** del Lenguaje DPsi. La etiqueta **Params** recibe los parámetros del diagrama a través de sus atributos. Cada atributo queda registrado en el diagrama y puede ser usado por cualquier elemento gráfico. La etiqueta **Bind** define los enlaces a las herramientas Psi de un elemento gráfico Psi. Tiene los siguientes lenguajes:

- Definición: `[[+|-|*] type=selector '|' event '|' key].`
- Métodos: `(+) open (-) close (*) destroy`

La etiqueta **Function** y **Make** tienen el mismo funcionamiento que las etiquetas **Function** y **Make** del lenguaje GLPsi Anexo B.5. La etiqueta **Include** adiciona un nuevo elemento al PsiData, tiene un alias (atributo *alias*) con el que se registra en PsiData, una url (atributo *url*) y el tipo de documento (atributo *type*) que puede ser un diagrama, una librería gráfica, fichero XML o un utilitario Psi. La etiqueta **Setting** modifica un elemento SVG de un elemento gráfico del diagrama, tiene una clave (atributo *key*) que corresponde a la clave del elemento SVG y la lista de atributos a modificar en ese elemento SVG. Por último, la etiqueta **Text** modifica el elemento **text** de SVG para crear textos multilínea.

FRAGMENTO B-5. Plantillas para las etiquetas Params, Include, Bind, Setting, Make y Function del Lenguaje DPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<Diagram xmlns:xlink="http://www.w3.org/1999/xlink" alias="…">
  <Params .../>
  <Bind .../>
  <Include alias="…" type="diagram|graph-library|document|tool" url="…" rewrite="yes|no"/> ...
  <Make id="…" ref="…"><![CDATA[ /*Cuerpo del Make*/ ]]></Make> ...
  <Function id="…" arguments="…"><![CDATA[ /*Cuerpo de la función*/ ]]></Function> ...
  <[Element]>
    <Bind ...="…"/>
    <Setting key="…" .../> ...
    <Text key="…" .../>  ...
    <Make id="…" ref="…"><![CDATA[ /*Cuerpo del Make*/ ]]></Make> ...
  </[Element]>
  ...
</Diagram>
```

En el FRAGMENTO B-6 se muestra la etiqueta **Skeleton** de un elemento gráfico. Tiene como objetivo simplificar la configuración de los elementos gráficos (Línea Psi, Figura Psi, Lista Psi y Contenedor Psi), es decir, es una abstracción que permite simplificar código de parametrización de elementos gráficos. Tiene un identificador único (atributo *id*) que es referenciado por el elemento gráfico. Tiene la referencia a un elemento de GLPsi (atributo *graph*), según el elemento gráfico se utilizan las etiquetas **Bind**, **Setting**, **Text**, **Make** y **Section**. En caso de que el elemento sea una línea, se pueden redefinir los rótulos inicial, medio y final.

FRAGMENTO B-6. Plantilla para la etiqueta Skeleton del Lenguaje DPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<Diagram xmlns:xlink="http://www.w3.org/1999/xlink" alias="...">
  <Skeleton id="…" graph="…"
            start-label="…" start-label-pos="…" start-label-class="…"
            middle-label="…" middle-label-pos="…" middle-label-class="…"
            end-label="…" end-label-pos="…" end-label-class="…">
    <Bind .../>
    <Setting key="…" .../> ...
    <Text key="…" .../>  ...
    <Make id="…" ref="…"><![CDATA[ /*Cuerpo del Make*/ ]]></Make>  ...
    <Section .../> ...
  </Skeleton>
  ...
</Diagram>
```

En el FRAGMENTO B-7 se muestra la etiqueta **Layer** y que se define el concepto *capa gráfica*. Esta etiqueta contiene otras capas (Etiqueta **Layer**) y elementos gráficos.

FRAGMENTO B-7. Plantilla para la etiqueta Layer del Lenguaje DPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<Diagram xmlns:xlink="http://www.w3.org/1999/xlink" alias="...">
  ...
  <Layer id="…">
    <Bind .../>
```

```
    <Bind .../>
    <Make id="…" ref="…"><![CDATA[ /*Cuerpo del Make*/ ]]></Make> ...
    <Layer id="…" .../> ...
    <[Element] id="…" .../> ...
  </Layer>
  ...
</Diagram>
```

B.7. Especificación DPsi de los Elementos Gráficos

En el FRAGMENTO B-8 se muestra la plantilla para las etiquetas **Icon**, **Node** y **Figure** representado con la etiqueta [**ElementShape**] del Lenguaje DPsi. Tiene un identificador (atribnuto *id*), posición (*x*,*y*) sobre el la *capa gráfica*, el redimensionamiento (*sx*, *sy*) y el ángulo de inclinación (atributo *angle*). Tiene la referencia ya sea a un elemento de GLPsi (atributo *graph*) o a un Skeleton (atributo *skeleton*). Las etiquetas **Bind**, **Setting** y **Make** se han explicado con anterioridad. La etiqueta **Text** crea un texto multilínea sobre un elemento **text** de SVG y se configuración según su ancho.

FRAGMENTO B-8. Plantillas para la etiqueta Icon, Node y Figure del Lenguaje DPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<Diagram xmlns:xlink="http://www.w3.org/1999/xlink" alias="...">
  ...
  <Layer id="…">
    <[ElementShape] id="…" x="…" y="…" sx="…" sy="…" angle="…" graph="…" skeleton="…">
      <Bind* ...="…"/>
      <Setting key="…" .../> ...
      <Text key="…" .../>  ...
      <Make* id="…" ref="…"><![CDATA[ /*Cuerpo de Make*/ ]]></Make> ...
      <Text key="…" .../> ...
    </[ElementShape]> ...
  </Layer> ...
</Diagram>
```

Nota: [ElementShape] = Icon | Node | Figure. *: Solo aplica a Node y Figura.

En el FRAGMENTO B-9 se muestra la plantilla para la etiqueta **Panel** del Lenguaje DPsi que representa un Contenedor de elementos gráficos de figuras, líneas, listas u otros contenedores (etiquetas [**Element**]). Según la Figura 5-11, el *lateral izquierdo* y *derecho* se defien mediante las etiquetas **Left** y **Right**, respectivamente. Además, cada *lateral* define un ancho (atributo *width*). Por otro lado, las etiquetas **Bind**, **Make** y **Setting** se han explicado en el anexo B.5.

FRAGMENTO B-9. Plantilla para la etiqueta Panel del Lenguaje DPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<Diagram xmlns:xlink="http://www.w3.org/1999/xlink" alias="...">
  ...
  <Layer id="…">
    <Panel id="…" x="…" y="…" sx="…" sy="…" width="…" height="…" angle="…" graph="…" skeleton="…">
      <Bind .../> ...
      <Make id="…" ref="…"><![CDATA[ /*Cuerpo de Make*/ ]]></Make> ...
      <Setting key="…" .../> ...
      <Text key="…" .../>  ...
      <Left width="…">
        <Make id="…" ref="…"><![CDATA[ /*Cuerpo de Make*/ ]]></Make> ...
        <[Element] id="…" .../> ...
      </Left>
      <Right width="…">
        <Make id="…" ref="…"><![CDATA[ /*Cuerpo de Make*/ ]]></Make> ...
        <[Element] id="…" .../> ...
      </Right>
      <[Element] id="…" .../> ...
```

```
    </Panel> ...
  </Layer> ...
</Diagram>
```

En el FRAGMENTO B-10 se muestra la plantilla para la etiqueta **List** del Lenguaje DPsi que representa el elemento gráfico Lista Psi de PsiDiagram. En la Figura 5-10, los ítems de una Lista Simple se definen mediante un selector jQuery (atributo *items*) de la etiqueta **List**. Para cada ítem su apariencia está definida en GLPsi. Para una Lista con Secciones se define la etiqueta **Section**, donde tiene los ítems (atributo *items*, selector jQuery) y un título de sección (atributo *header*).

FRAGMENTO B-10. Plantilla para la etiqueta **List** del Lenguaje DPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<Diagram xmlns:xlink="http://www.w3.org/1999/xlink" alias="...">
  ...
  <Layer id="…">
    <List id="…" x="…" y="…" sx="…" sy="…" width="…" angle="…" graph="…" skeleton="…" items="…">
      <Bind .../> ...
      <Text key="…" .../>  ...
      <Make id="…" ref="…"><![CDATA[ /*Cuerpo de Make*/ ]]></Make> ...
      <Section header="…" items="…" .../> ...
    </List> ...
  </Layer> ...
</Diagram>
```

En el FRAGMENTO B-11 se muestra las plantillas de las etiquetas **Line** y **Connector** del Lenguaje DPsi que implementa un Linea Psi. Los rótulos inicial, medio y final de una Línea Psi atributos [labels], son descritos en la sección 5.5.1.1. Para la etiqueta **Connector** se definen los elementos gráficos que conecta (atributos *from* y *to*) y en que punto del borde de conexión (atributos *start* y *end*).

FRAGMENTO B-11. Plantillas para las etiqueta **Line** y **Connector** del Lenguaje DPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<Diagram xmlns:xlink="http://www.w3.org/1999/xlink" alias="...">
  ...
  <Layer id="…">
    <[ElementPath] id="…" x="…" y="…" sx="…" sy="…" width="…" angle="…" graph="…" skeleton="…"
                   start="…" from="…" end="…" to="…" [labels]>
      <Bind .../> ...
      <Text key="…" .../>  ...
      <Make id="…" ref="…"><![CDATA[ /*Cuerpo de Make*/ ]]></Make> ...
    </[ElementPath]> ...
  </Layer> ...
</Diagram>
```

Nota: [ElementPath] = Line | Connector.

B.8. Especificación TPsi de elementos de Contenido, Posición, Eventos y Opciones

En el FRAGMENTO B-12 se muestran las plantillas genéricas **[Options]**, **[Container]**, **[Position]** y **[Events]** para las herramientas del Lenguaje TPsi.

La plantilla [Options] especifica las etiquetas **Options** y **Var** para una herramienta. La etiqueta **Options** es una lista de opciones como títulos, rótulos, colores, etc., para personalizar una herramienta. Además, construye un objeto JavaScript y sus atributos son campos de este objeto, donde el nombre de atributo es el nombre del campo mientras que el valor de atributo es el valor del campo. La etiqueta anidada **Options**, define un nuevo objeto cuyo nombre de campo es el atributo *key*. Por otro lado, la etiqueta **Var** define variables a usar en las herramientas, normalmente son utilizadas por los eventos de la herramienta.

La plantilla [Container] especifica las etiquetas **XslParams** y **Content** para una herramienta. La etiqueta **Content** implementa el contenido de una herramienta, ya sea de tipo textual, fragmento de páginas, imágenes y transformaciones XML con XSLT (atributo *type*). Las transformaciones requieren de los documentos XML (atributo *data-xml* de **Content**) y XSLT (se definen con la etiqueta **XslParams**).

La plantilla [Position] especifica la etiqueta **Position** de una herramienta. Esta etiqueta define la posición de salida con respecto a un elemento gráfico o un elemento HTML y sigue las especificaciones de *position* de jQueryUI (jqueryui.com/position).

Por último, la plantilla [Events] especifica múltiples etiquetas **Event** para una herramienta. Una etiqueta **Event** define un evento de una herramienta. El conjunto de eventos básicos propios de todas las herramientas son: *onCreate*, *onDestroy*, *onOpen*, *onClose* y *onRefresh*. Cada evento se implementa en JavaScript y tiene su lista de argumentos (atributo *arguments*).

FRAGMENTO B-12. Plantillas para las etiquetas [ElementOptions], [ElementContainer], [ElementEvents] y [ElementPosition] del Lenguaje TPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<ToolsPsi id="…" alias="…" module="…">
  ...
  <[Options] id="…" ...>
    <Options ...>
      <Options key="…" .../> ...
    </Options>
    <Var id="" type="string|number|integer|object|xml|jxml|jQuery|map"
         public="yes|no" description="…"/> ...
  </[Options]>

  <[Container] id="…" ...>
    <XslParams _data-document="…" _data-url="…"/>
    <Content type="text|xml|fragment|image|transform" reload="no|yes"/>
  </[Container]>

  <[Position] id="…" ...>
    <Position my="…" at="…" of="…" collision="…"/>
  </[Options]>

  <[Events] id="…" ...>
    <Event name="…" arguments="…" /> ...
  </[Events]>
  ...
</ToolsPsi>
```

B.9. Especificación TPsi de las Herramientas Visuales

En el FRAGMENTO B-13 se muestra la plantilla para las etiquetas **Toolbar**, **Tooltip** y **Dialog**. Estas etiquetas tienen el identificador (atributo *id*), la herencia de herramienta (atributo *inherited*), la clase CSS (atributo *class*) y tipo de cerrado (atributo *closed*). Además, estas etiquetas definen etiquetas genéricas [Options], [Container], [Position] y [Events] presentadas en el Anexo B.8.

En la etiqueta **Dialog** se define el conjunto de botones mediante múltiples etiquetas **Button**. Una etiqueta **Button** implementa la lógica en JavaScript de un botón en la parte inferior de la caja de diálogo, define el rótulo del botón (atributo *name*) y la lista de argumentos (atributo *arguments*).

FRAGMENTO B-13. Plantillas para las etiquetas Toolbar, Tooltip y Dialog del Lenguaje TPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<ToolsPsi id="…" alias="…" module="…">
  ...
  <Toolbar id="…" inherits="…" class="…" close="auto|none">
```

```
    [Options][Container][Position][Events]
  </Toolbar>
  ...
  <Tooltip id="…" inherits="…" class="…" close="auto|none">
    [Options][Container][Position][Events]
  </Tooltip>
  ...
  <Dialog id="…" inherits="…" class="…" close="auto|none">
    [Options][Container][Position][Events]
    <Button name="…" arguments="…"/> ...
  </Dialog>
  ...
</ToolsPsi>
```

En el FRAGMENTO B-14 se muestra la plantilla para las etiquetas **Drop** y **DragDrop**. Estas etiquetas tienen en común el identificador (atributo *id*), la referencia a una etiqueta **Var** que define una plantilla de un elemento gráfico a adicionar en un programa DPsi (etiqueta *map*), el tipo de refresco (atributo *refresh*), el selector jQuery en donde se adiciona el elemento gráfico (atributo *append-selector*) y el tipo de ocultación (atributo *hide*). Adicionalmente, la etiqueta **DragDrop** define un selector padre para elegir los elementos a arrastrar (atributo *parent-selector*). Además, estas etiquetas definen las etiquetas genéricas [Options], [Container] y [Events] presentadas en el Anexo B.8.

En la etiqueta **Drop** se define la etiqueta **FrameDrop** la cual define el Contenedor donde se permite soltar por medio de un selector jQuery (atributo *selector*) y define un **div** de HTML y una apariencia (atributo *class*).

En la etiqueta **DragDrop** se defienen las etiquetas **FrameDrag** y **FrameDrop**. La etiqueta **FrameDrag** define el/los contenedor(es) en los cuales se puede iniciar el arrastre mediante un selector jQuery (atributo *selector*) y define un **div** de HTML y una apariencia (atributo *class*). La etiqueta **FrameDrop** es la misma definida en la etiqueta **Drop**.

FRAGMENTO B-14. Plantillas para las etiquetas Drop y DragDrop del Lenguaje TPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<ToolsPsi id="…" alias="…" module="…">
  ...
  <Drop id="…" map="…" refresh="always|none" append-selector="…" hide="auto|none">
    [Options][Container][Events]
    <FrameDrop selector="…" class="…">
      <Options key="…" .../> ...
    </FrameDrop>
  </Drop>
  ...
  <DragDrop id="…" map="…" parent-selector="…" body="…" append-selector="…" hide="auto|none">
    [Options][Container][Events]
    <FrameDrag selector="…" class="…">
      <Options key="…" .../> ...
    </FrameDrag>
    <FrameDrop selector="…" class="…">
      <Options key="…" .../> ...
    </FrameDrop>
  </DragDrop>
  ...
</ToolsPsi>
```

En el FRAGMENTO B-14 se muestra la plantilla para las etiquetas **Menu** y **Popup**. Estas etiquetas tienen en común el identificador (atributo *id*), el formato del menú con clases CSS (atributo *class*) y el tipo de cerrado (atributo *closed*). Adicionalmente, la etiqueta **DragDrop** define un selector padre para elegir los elementos a arrastrar (atributo *parent-selector*). Estás etiquetas definen las etiquetas genéricas

[Options], y además, la etiqueta **Popup** define la etiqueta [Position] y [Events] presentadas en el Anexo B.8.

Para **Menu** y **Popup** se definen las etiquetas **Item**, **Space** y **Submenu**. La etiequeda **Item** especifica un ítem de menú, tiene un identificador (atributo *id*), un rótulo (atributo *label*), una clase CSS (atributo *class*), un ícono (atributo *icon*), una fuente de letra (atributo *font*) y una referencia a una herramienta (atributos *type* y *tool*). La etiqueta **Space** crea una división entre ítems o submenús. Por último, la etiqueta **Submenu** define un submenús de ítems o submenús, tiene un identificador (atributo *id*), un rótulo (atributo *label*), una clase CSS (atributo *class*) y un ícono (atributo *icon*).

FRAGMENTO B-15. Plantillas para las etiquetas Menu y Popup del Lenguaje TPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<ToolsPsi id="…" alias="…" module="…">
  ...
  <Menu id="…" class="…" close="auto|none" design="button|toolbar|list">
    [Options]
    <Item id="…" label="…" type="…" active="false|true" active-class="…" reload="no|yes" tool="…"
          key="…" text="false|true" icon="…" font="…" enable="false|true"/> ...
    <Space width="..." height="..." hr="yes|no"/> ...
    <Submenu id="…" label="…" class="…" label-class="…" icon="…" font="…" collapse="yes|no">
      <Submenu .../> ...
    </Submenu> ...
  </Menu>
  ...
  <Popup id="…" class="…" close="auto|none" design="button|toolbar|list">
    [Options][Position]
    <Item id="…" label="…" type="…" active="false|true" active-class="…" reload="no|yes" tool="…"
          key="…" text="false|true" icon="…" font="…" enable="false|true"/> ...
    <Space width="…" height="…" hr="yes|no"/> ...
    <Submenu id="…" label="…" icon="…" font="…">
      <Submenu .../> ...
    </Submenu>
  </Popup>
  ...
</ToolsPsi>
```

B.10. Especificación TPsi de las Utilidades de Programación

En el FRAGMENTO B-16 se presentan las plantillas para las etiquetas **LoadDocument**, **Script**, **Make**, **Function**, **Var**, **Class** y **Command**.

La etiqueta **LoadDocument** define cómo cargar un documento en el PsiData mediante una URL (atributo *url*) y se designa un alias (atributo *alias*). Con la etiqueta **Script** se pueden definir bloques de código JavaScript, mientras que la etiqueta **Make** implementa un modificador de elementos gráficos para el lenguaje DPsi como herramienta.

La etiqueta **Command** define un comando de ejecución en lenguaje JavaScript utilizado en el lenguaje DPsi para invocar diferentes herramientas desde eventos programados en los elementos gráficos. Contiene la etiqueta **Body** (cuerpo del comando en JavaScript) y las etiquetas genéricas [Options] presentadas en el Anexo B.8.

El lenguaje TPsi un conjunto de elementos de programación básico **Function**, **Var** y **Class**. La etiqueta **Function** se implementa una función JavaScript, la etiqueta **Var** implementa una variable JavaScript y la etiqueta **Class** implementa una clase con propiedades (etiqueta **Properties**) y métodos (múltiples etiquetas **Method**).

FRAGMENTO B-16. Plantillas para las etiquetas **LoadDocument**, **Script**, **Make**, **Function**, **Var**, **Class** y **Command** del Lenguaje TPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<ToolsPsi id="…" alias="…" module="…">
  ...
  <LoadDocument alias="..." url="..." rewrite="yes|no"/> ...
  <Script id="..."><![CDATA[
    //Contenido del script
  ]]></Script> ...
  <Make id="..."><![CDATA[
    //Cuerpo del Make
  ]]></Make> ...
  <Function id="..." arguments="..."><![CDATA[
    //Cuerpo de la función
  ]]></Function> ...
  <Var id="..." type="string|number|integer|object|xml|jxml|jQuery|map"
       public="yes|no" description="..."/>
  <Class name="..." module="..." inherits="..." init="...">
    <Properties .../>
    <Method name="..." arguments="..."><![CDATA[
      //Cuerpo del método
    ]]></Method> ...
  </Class> ...
  <Command id="...">
    [Options]
    <Body name="..." arguments="..."><![CDATA[
      //Cuerpo del comando
    ]]></Body>
  </Command>
  ...
</ToolsPsi>
```

B.11. Especificación TPsi del Generador de Diagrama

En el FRAGMENTO B-17 se muestra la plantilla para generación de diagramas, etiqueta **Generator**. Esta etiqueta tiene un identificador (atributo *id*), un nombre (atributo *name*), un módulo (atributo *module*), el generador padre (atributo *inherited*) y una inicialización (atributo *init*). Además, define las etiquetas **Graphs**, **Lines**, **Process**, **DefinitionMatrix** y las etiquetas genéricas [Options] presentadas en el Anexo B.8.

La etiqueta **Graphs** especifica la generación de elementos gráficos figuras, listas y contenedores para el lenguaje DPsi. Tiene un identificador (atributo *id*), un tipo de posición (atributo *type*, aleatorio o fijo), una referencia a una capa gráfica (atributo *append-to*) y una consulta jQuery sobre un documento XML para obtener los elementos a generar (atributo *ref-document*). La etiqueta **Source** es una plantilla para especificar una figura, lista o contenedor de DPsi y tiene las etiquetas genéricas [Events] presentadas en el Anexo B.8.

La etiqueta **Lines** especifica la generación de líneas para el lenguaje DPsi. Tiene un identificador (atributo *id*), una referencia a una capa gráfica (atributo *append-to*), una consulta jQuery sobre un documento XML para obtener la información asociada a las líneas a generar (atributos *to*, *from* y *ref-document*). La etiqueta **Source** es la plantilla para especificar un conector de DPsi y tiene las etiquetas genéricas [Events] presentadas en el Anexo B.8.

La etiqueta **Process** implementa los algoritmos de procesamiento de los elementos gráficos para modificar los elementos gráficos creados. Tiene un nombre (atributo *name*) y unos argumentos (atributo *arguments*) y el cuerpo del proceso escrito en JavaScript.

Por último, la etiqueta **DefinitionMatrix**, define una matriz bidimensional (atributos *cols*, *rows*) para facilitar la creación de algoritmos. Se puede definir el ancho y alto de las celdas de la matriz (atributos *cell-width* y *cell-height*, respectivamente) o la proporción del elemento gráfico (atributos *graph-width* y *graph-height*).

FRAGMENTO B-17. Plantillas para la etiqueta **Generator** del Lenguaje TPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<ToolsPsi id="…" alias="…" module="…">
  ...
  <Generator id="…" name="…" module="..." inherits="..." init="...">
    [Options]
    <Graphs id="…" position="random|fixed" append-to="…" ref-document="…" attached-to="…">
      <Source><![CDATA[ ... ]]></Source>
      [Events]
    </Graphs> ...
    <Lines id="…" from="…" to="…" append-to="…" ref-document="…" path-itself="…">
      <Source><![CDATA[ ... ]]></Source>
      [Events]
    </Lines> ...
    <Process name="…" arguments="…"><![CDATA[
      //Cuerpo del proceso de generación
    ]]></Process> ...
    <DefinitionMatrix id="…" cols="…" rows="…" top="…" left="…" empty="…" scale="…" graph-size="…"
                      graph-width="…" graph-height="…" cell-width="…" cell-height="…"/> ...
  </Generator>
  ...
</ToolsPsi>
```

Anexo C. Especificaciones PsiGVA

C.1. Especificación PsiGVA para la gramática GLPsi

En la Tabla C-1 se muestran los validadores de atributos de las etiquetas **GraphsLibrary**, **Function**, **Make**, **ConnectionEdge**, **Edge**, **Shape**, **Container**, **List**, **Line**, **Events**, **SourceSVG**, **IconSVG**, **Settings**, **LineSetting** y **MarkSetting**, para la estructura de la gramática GLPsi.

Tabla C-1. Campo VALIDATOR de las estructura de datos para la Gramática GLPsi.

Gramática GLPsi	
Etiqueta	**VALIDATOR**
GraphsLibrary	*alias:text
Function	*identifier|arguments:text
Make	*identifier
ConnectionEdge	*id:identifier|default:text
Edge	*id:identifier|*type:select(points,segment)|*source:text| class:text|discretize=8:number(5,8,10,16,20)
Shape	*id:identifier|connection:identifier|midpoint=X+W/2,Y+H/2:text(X+W/2@#Y+H/2,W/2@#H/2)|before-key-select:text
Container	*id:identifier|connection:identifier|midpoint=X+W/2,Y+H/2:text(X+W/2@#Y+H/2,W/2@#H/2)|before-key-select:text
List	*id:identifier|connection:identifier|midpoint=X+W/2,Y+H/2:text(X+W/2@#Y+H/2,W/2@#H/2)|before-key-select:text
Line	*id:identifier|minimum=none:select(none,own,both,other)|duplicate=yes:select(yes,no)|midpoint=0.5: number;range(0,1)
Events (a)	drag:text(selector)|select:text|hide:text|transform:text|toggle-connection:text
Events (b)	checkpoints:text|select:text|drag:text
SourceSVG	id:identifier|url:text
IconSVG	id:identifier|url:text
Settings (a)	scale-min=0.25:number|scale-max=2:number|scale-r=5:number|scale-class=glpsi-transform-scale:identifier|box-class=glpsi-transform-box:identifier
Settings (b)	width=300:number|width-left=0:number|width-right=0:number|height=300:number|body-class=glpsi-container-body:text|background-class=glpsi-container-background:text|background-padding=3:number|background-rx=0:number|background-ry=0:number|space-header=3:number|space-footer=3:number|space-lateral=3:number
Settings (c)	width=150:number|background-class=glpsi-list-background:text|background-padding=3:number|background-rx=4:number|background-ry=4:number|space-header=3:number|space-item=4:number|divider-class=glpsi-list-divider:text
LineSetting	class:text|fill=none:text|stroke=gray:text|stroke-width=1.5px:text|marker-start:text|marker-end:text
MarkSetting	fill=#ffffe8:text|stroke=gray:text|stroke-width=0.3px:text|r=4:number

Nota: Script, Defs, Commands, Text, HeaderSVG, FooderSVG, SctionSVG, ItemSVG, OnResize, OnSection y OnItem son nulos.

Las etiquetas Script, Defs, Commands, Text, HeaderSVG, FooderSVG, SctionSVG, ItemSVG, OnResize, OnSection y OnItem no tienen definido validadores.

C.2. Especificación PsiGVA para la gramática DPsi

La especificación de los validadores de atributos PsiGVA para la gramática GLPsi se muestra en la siguiente tabla:

Tabla C-2. Campo VALIDATOR de la estructura de datos para la Gramática DPsi.

Gramática DPsi	
Etiqueta	**VALIDATOR**
Diagram	*id:identifier|*alias:text|width:number|height:number|scale:number
Params	type:text|theme:text
Include	*alias:text|*type:select(diagram,graph-library,document,tool)|*url:text| rewrite=no:select(yes,no)
Function	*id:identifier|arguments:text
Make	id:identifier|ref:identifier
Skeleton	*id:identifier|*graph:identifier|start-label:text|start-label-pos:text|start-label-class:text|middle-label:text|middle-label-pos:text|middle-label-class:text|end-label:text|end-label-pos:text|end-label-class:text
Layer	*id:identifier
Icon	*id:identifier|*x:number|*y:number|sx=1:number|sy=1:number|angle=0:number|graph:identifier|skeleton:identifier
Node	*id:identifier|*x:number|*y:number|sx=1:number|sy=1:number|angle=0:number|graph:identifier|skeleton:identifier
Figure	*id:identifier|*x:number|*y:number|sx=1:number|sy=1:number|angle=0:number|graph:identifier|skeleton:identifier
Panel	*id:identifier|*x:number|*y:number|*width:number|*height:number|sx=1:number|sy=1:number|graph:identifier|skeleton:identifier
Lateral	*width:number
List	*id:identifier|*x:number|*y:number|*width:number|sx=1:number|sy=1:number|angle=0:number|items:text|graph:identifier|skeleton:identifier
Section	*key:identifier|class:text
Line	*id:identifier|graph:identifier|skeleton:identifier|*start:text|*path:text|*end:text|class:text
Connector	*id:identifier|graph:identifier|skeleton:identifier|from:identifier|*start:text|*path:text|to:identifier|*end:text|class:text
Bind	help.operator:text|help.type:text|help.command:text
Setting	*key:identifier
Text	*key:identifier

C.3. Especificación PsiGVA para la gramática TPsi

La especificación de los validadores de atributos PsiGVA para la gramática TPsi se divide en dos tablas: en la Tabla C-3 se muestra el PsiGVA para las Herramientas Visuales, mientras que en la

Tabla C-3. Campo VALIDATOR de las estructura de datos de la Gramática TPsi (Herramientas Visuales).

Gramática TPsi	
Etiqueta	**VALIDATOR**
ToolsPsi	*alias:identifier|module:identifier
Var	*id:identifier|type=string:select(string,number,integer,object,xml,jxml,jQuery,map)|public:text(yes,no)|description:text

Dialog	*id:identifier|close=none:select(auto,none)|inherits:text|class:text
Toolbar	*id:identifier|close=none:select(auto,none)|inherits:text|class:text
Tooltip	*id:identifier|close=auto:select(auto,none)|inherits:text|class:text
Frame	*id:identifier|close=none:select(auto,none)|inherits:text|class:text
Drop	*id:identifier|close=auto:select(auto,none)|map:identifier|parent-selector=body:text|append-selector:text|hide=auto:select(auto,none)
DragDrop	*id:identifier|close=auto:select(auto,none)|map:identifier|parent-selector=body:text|append-selector:text|hide=auto:select(auto,none)
Popup	*id:identifier|close=auto:select(auto,none)|design=popup:select(popup)
Menu	*id:identifier|close=none:select(auto,none)|design=button:select(button,toolbar,list)
Space	width=20px:text(10px,20px)|height=10px:text(10px,20px)|hr=no:select(yes,no)
Options	height:text|width:text|model:text(true,false)|title:text|arrow:text|arrowLeft:text
XslParams	_data-document:text|_data-url:text
Position	my:text('left top','left center','left bottom','center top','center center','center bottom','right top','right center','right bottom')|at:text('left top','left center','left bottom','center top','center center','center bottom','right top','right center','right bottom')|of:text(window)|collision:text('flip flip','fit fit','flipfit flipfit', 'none none')
Content	*type=text:select(text,xml,fragment,image,transform)|reload=no:select(no,yes)
Event	*name:identifier(--Events Base--,beforeOpen,afterOpen,refresh,close,--Events UI jQuery--,create,open,close,beforeClose,drag,dragStart,dragStop,focus,resize,resizeStart,resizeStop)|arguments:text(--Events Base--,container@# proxy,--Events UI jQuery--,event@# ui,event@# ui@#self)
Button	name:text|arguments:text(event@# ui)
SubMenu(a)	*label:text|icon:text|font:text
SubMenu(b)	*label:text|class:classcss|label-class:classcss|icon:text|font:text|collapse=yes:select(yes,no)
Item (a)	*id:identifier|*label:text|*type:identifier|active=false:boolean|active-class:classcss|reload=no:select(no,yes)|*tool:text|key=default:text|icon:text|font:text|enable=true:select(true,false)
Item (b)	*id:identifier|*label:text|*type:identifier|active=false:boolean|active-class:classcss|reload=no:select(no,yes)|*tool:text|key=default:text|text=true:select(true,false)|icon:text|font:text|enable=true:select(true,false)

Tabla C-4. Campo VALIDATOR de las estructura de datos de la Gramática TPsi (Utilidades de Programación).

Gramática TPsi	
Etiqueta	**VALIDATOR**
ToolsPsi	*alias:identifier|module:identifier
LoadDocument	*alias:text|*url:text|rewrite=no:select(yes,no)
Var	*id:identifier|type=string:select(string,number,integer,object,xml,jxml,jQuery,map)|public:text(yes,no)|description:text
Make	*id:identifier
Script	*id:identifier
Function	*id:identifier|arguments:text
Class	*name:identifier|module:identifier|inherits:identifier|init:text
Method	*name:identifier|arguments:
Command	*id:identifier
Dialog	*id:identifier|close=none:select(auto,none)|inherits:text|class:text
Generator	*id:identifier|manager-diagram:identifier
Graphs	*id:identifier|position=random:select(random,fixed)|append-to:text(selector)|ref-document:text|attached-to:text(fromId,toId)
Lines	*id:identifier|from:text|to:text
Process	*name:identifier|arguments:text
DefinitionMatrix	*id:identifier|cols=10:number|rows=10:number|top=0:number|left=0:number|empty=0.5:number|scale=1:number|graph-size=25:number|graph-width=100:number|graph-height=50:number|cell-width=1.5:number|cell-height=1.5:number

Nota: Properties, Body y Source son nulos.

C.4. Especificación PsiGVA para la gramática BPMEPsi

La especificación de los validadores de atributos PsiGVA para la gramática BPMEPsi se muestra en la Tabla C-5.

Tabla C-5. Campo VALIDATOR de las estructura de datos de la Gramática BPMEPsi.

Gramática BPMEPsi	
Etiqueta	**VALIDATOR**
BPMExecutionPsi	*start-process:integer
Process	*id:identifier
StartEvent	*id:integer|*type:identifier|name:text
IntermediateEvent	*id:integer|*type:identifier|name:text
EndEvent	*id:integer|*type:identifier|name:text
Gateway	*id:integer|*type:identifier|name:text
Task	*id:integer|*type:identifier|name:text
TaskEvent	*id:integer|*type:identifier
Flow	*id:integer|*type:identifier|name:text|*source:integer|*target:integer
Data	*id:integer|*type:identifier|name:text

Anexo D. Marcos de Página Web Psi

Este anexo define el concepto de Marco de Página Web Psi que hace parte del diagrama programable (PsiDiagram). Para ello se crea el **Lenguaje WPPsi** que define una página web dividida por secciones, y cada sección es cargada y actualizada con diferentes tipos de contenido. Ayuda al diseño e implementación de DSVL, diagramas y generación de diagramas en un CSWE (O3) basados en el PsiEngine (O1). Este anexo se divide en tres partes: definición conceptual (sección D.1), especificación del lenguaje WPPsi (sección D.2) e implementación del componente WPPsi (sección D.3).

D.1. Definición Conceptual

En la Figura D-1 se presenta una página web distribuida por secciones denominado **Marco de Página**. Una **Sección** es una lista de *contenidos*. Un **Contenido** es un fragmento de página web creado dinámicamente o cargado como recursos desde el servidor web. Un Contenido puede ser: fragmentos HTML, transformaciones XML+XSLT, Barras de Herramientas Psi, Menú Psi, Marco de Contenido Psi, Diagrama Psi y Widget de jQuery. Pueden existir diferentes Tipos de Secciones y en el lado izquierdo de la Figura D-1 se presentan las más usadas en el diseño de páginas web. Además, cada Contenido debe tener la capacidad de cargarse, refrescarse y eliminarse.

En el ejemplo de Figura D-1 la página web está distribuida en tres secciones: *cabecera*, *sección lateral* y *sección central*. En la *cabecera* hay un marco de contenido y una barra de herramientas Psi, en la *sección lateral* hay un menú Psi, mientras en *sección central* hay un diagrama Psi y un marco de contenido.

Con la definición de Marco de Página se pueden crear aplicaciones web de una página. Para ello se requiere tener un conjunto de Marcos de Página denominado **Marcos de Página Web Psi** y poder navegar entre ellos. Cada Contenido puede ser usado en más de una Sección y un Marco de Página. Además, cada Contenido debe tener la facilidad de intercambio de información con otros contenidos.

En resumen, los Marcos de Página Web Ps**i** deben tener la capacidad de ofrecer:

- Definición de Marco de Página.
- Asociación y manipulación de fuentes de información.
- Inclusión de Herramientas Psi y Widgets jQuery.

Los Marcos de Páginas Psi son especificados con el **Lenguaje WPPsi**. El lenguaje WPPsi es implementado en el **Componente WPPsi**.

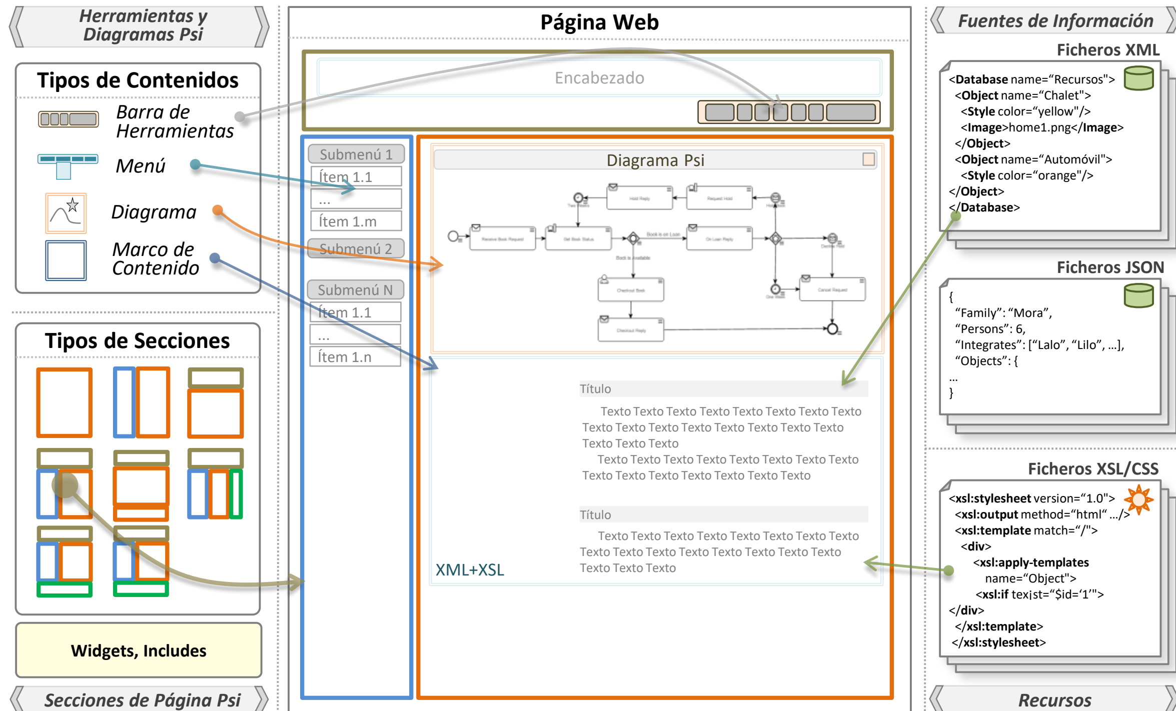
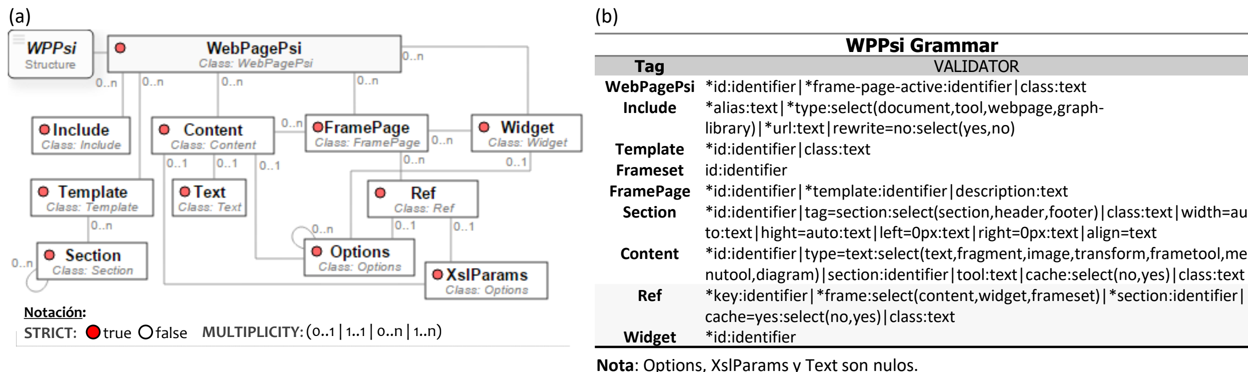


Figura D-1. Plantilla de diseño por secciones de una página web.

D.2. Especificación del Lenguaje WPPsi

El **Lenguaje WPPsi** especifica un Marco de Página y Secciones para diseñar una página web. Este lenguaje tiene la capacidad de interactuar con el Kernel de **PsiDiagram**. Ayuda a facilitar la construcción de modelos visuales facilitando la interacción con la Diagramación Psi, las Herramientas Visuales y las Herramientas de Programación Psi.

En la Figura D-2(a) se muestra el PsiLSD del Lenguaje WPPsi y en la Figura D-2(b) se muestra el PsiGVA para la gramática WPPsi. La etiqueta raíz **WebPagePsi** es la etiqueta raíz para el lenguaje WPPsi. Contiene un identificador (atributo *id*), el marco de página activo (atributo *frame-page-active*) y una clase CSS (atributo *class*). La etiqueta raíz contiene las etiquetas **Include**, **Template**, **Content**, **FramePage** y **Widget**.



WPPsi Grammar

Tag	VALIDATOR
WebPagePsi	*id:identifier|*frame-page-active:identifier|class:text
Include	*alias:text|*type:select(document,tool,webpage,graph-library)|*url:text|rewrite=no:select(yes,no)
Template	*id:identifier|class:text
Frameset	id:identifier
FramePage	*id:identifier|*template:identifier|description:text
Section	*id:identifier|tag=section:select(section,header,footer)|class:text|width=auto:text|hight=auto:text|left=0px:text|right=0px:text|align=text
Content	*id:identifier|type=text:select(text,fragment,image,transform,frametool,menutool,diagram)|section:identifier|tool:text|cache:select(no,yes)|class:text
Ref	*key:identifier|*frame:select(content,widget,frameset)|*section:identifier|cache=yes:select(no,yes)|class:text
Widget	*id:identifier

Nota: Options, XslParams y Text son nulos.

Figura D-2. (a) El PsiLSD para el lenguaje WPPsi. (b) El PsiGVA de la Gramática WPPsi.

En el FRAGMENTO D-1 se muestra la plantilla para la etiqueta **Content** que implementa el Contenido de un Marco de Página. Tiene un identificador único (atributo *id*), el tipo de contenido como texto, fragmentos HTML, imágenes, transformaciones XML+XSLT, Marco de Contenido Psi, Barras de Herramientas Psi, Menú Psi y Diagrama Psi (atributo *type*), la referencia a una herramienta (atributo *tool*), una clase CSS (atributo *class*), la referencia a una Sección (atributo *section*) y si puede guardarse en memoria caché (atributo *cache*). Contiene las etiquetas **Options**, **XslParams** y **Text** y tienen la misma funcionalidad que las etiquetas de la Herramienta Marco de Página Psi. Por otro lado, la etiqueta **Widget** implementa el llamado a cualquier widget jQuery UI (http://api.jqueryui.com/) que es otra forma de Contenido.

FRAGMENTO D-1. Plantillas para las etiquetas **Content** y **Widget** del Lenguaje WPPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<WebPagePsi id="…" xmlns:xlink="…" frame-page-active="…" class="…">
  ...
  <Content id="…" type="text|fragment|image|transform|frametool|menutool|diagram"
           section="…" tool="…" cache="yes|yes" class="…">
    <Options ...>
      <Options .../> ...
    </Options>
    <XslParams .../>
    <Text .../>
  </Content> ...
  <Widget id="…" class="…" section="…" >
    <Options ...>
      <Options .../> ...
    </Options>
  </Widget> ...
  ...
</WebPagePsi>
```

En el Fragmento D-2 se presentan las plantillas para las etiquetas **Include** y **Template**. La etiqueta **Include** facilita la inclusión de Librerías Gráficas Psi, Herramientas Psi y Documentos XML con la ayuda de los atributos *type* y *url*, asignando un alias (atributo *alias*).

La etiqueta **Template** es la que define los Tipos de Secciones. Esta etiqueta tiene un identificador (atributo *id*) y un clase CSS (atributo *class*) y contiene múltiples etiquetas **Section**. La etiqueta **Section** define una Sección de un Marco de Página. Esta etiqueta determina el tipo de sección (atributo *type*), su alineación (atributo *align*), su ancho (atributo *width*), su alto (atributo *height*), su margen izquierda (atributo *left*) y su margen derecha (atributo *right*), además puede contener secciones anidadas. Una etiqueta **Template** puede estar en más de un Marco de Página.

Fragmento D-2. Plantillas para las etiquetas **Include** y **Template** del Lenguaje WPPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<WebPagePsi id="…" xmlns:xlink="…" frame-page-active="…" class="…">
  ...
  <Include alias="…" type="document|tool|webpage|graph-library" url="…" rewrite="yes|no"/> ...
  <Template id="…" class="…">
    <Section id="…" type="section|header|footer" class="…" align="none|justify|left|center|right"
             width="…" hight="…" left="…" right="…">
      <Section id="…" ...="…" .../> ...
    </Section> ...
  </Template>
  ...
</WebPagePsi>
```

Por último, en el Fragmento D-3 se muestra la plantilla de la etiqueta **FramePage**. Esta etiqueta implementa el concepto de Marco de Página. Tiene un identificador (atributo *id*), la referencia a la

plantilla de secciones a utilizar (atributo *template*) y una descripción (atributo *description*). Contiene las etiquetas **Content, Widget** y **Ref**. Las etiquetas **Content y Widget** han sido descritas en el FRAGMENTO D-1, y se debe añadir en qué sección se incluye (atributo *section*). La etiqueta **Ref** es una referencia a Contenidos y Widgets definidos con anterioridad (atributo *key*), el tipo de marco (atributo *frame*) la sección a la que se añade (atributo *section*), una clase CSS (atributo *class*) y si es cacheable (atributo *cache*). Cabe notar la simplicidad de construir Marcos de Página, de que solo se incluye su lista de Contenidos.

Fragmento D-3. Plantilla para la etiqueta **FramePage** del Lenguaje WPPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<WebPagePsi id="…" xmlns:xlink="…" frame-page-active="…" class="…">
  ...
  <FramePage id="…" template="…" description="…">
    <Content id="…" section="…" .../>  ...
    <Widget id="…" section="…" .../>  ...
    <Ref key="…" frame="content|widget" section="…" cache="no|yes" class="…">
      <Options ...="…" .../>
      <XslParams ...="…" .../>
    </Ref> ...
  </FramePage>
  ...
</WebPagePsi>
```

D.3. Componente WPPsi

En la Figura D-3 se muestra el diagrama de clases del **Componente WPPsi**. Este componente implementa la funcionalidad del Marcos de Página Web Psi.

La clase *WebPagePsi* implementa la clase raíz del lenguaje WPPsi y contiene la lista de Marcos de Página (atributo *frames*, múltiples instancias de *FramePage*), lista de Contenidos (atributo *frames*, múltiples instancias de *Content*), lista de widgets (atributo *widgets*, múltiples instancias de *Widget*), la lista de Tipos de Secciones (atributo *templates*, múltiples instancias de *Template*) y la lista de inclusiones (atributo *includes*, múltiples instancias de *Include*).

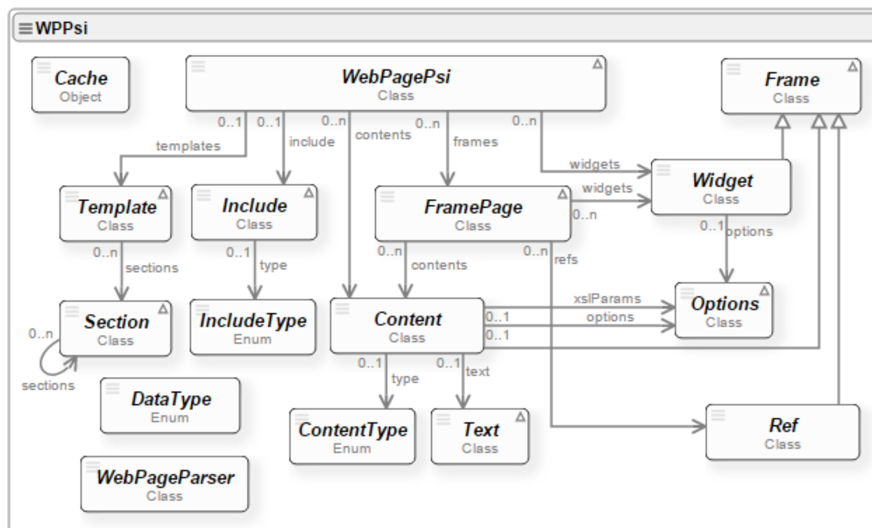


Figura D-3. Diagrama de clases del Componente WPPsi.

La clase *Frame* es la base para cualquier tipo de contenido. La clase *Content* implementa el Contenido y tiene la funcionalidad de cargar fragmentos de página, transformaciones XSLT y Herramientas Psi, mientras que la clase *Widget* implementa la funcionalidad para llamar cualquier Widget creado como plugin de jQueryUI. La clase *Ref* implementa los llamados a Contenidos y Widgets.

La clase *Template* define los Tipos de Secciones. Tiene una lista de secciones (atributo *sections*, múltiples instancias de *Section*) y crea un mapa para su búsqueda de secciones. La clase *Section* implementa la lógica para crear una sección utilizando la etiqueta **div** de HTML.

La clase *Include* implementa la lógica para incluir Librerías Graficas Psi, Herramientas Psi y Documentos XML. El objeto *Cache* administra los contenidos que pueden ser guardados en memoria caché, de tal forma que guarda la referencia a este contenido y cuando se solicita se muestran nuevamente.

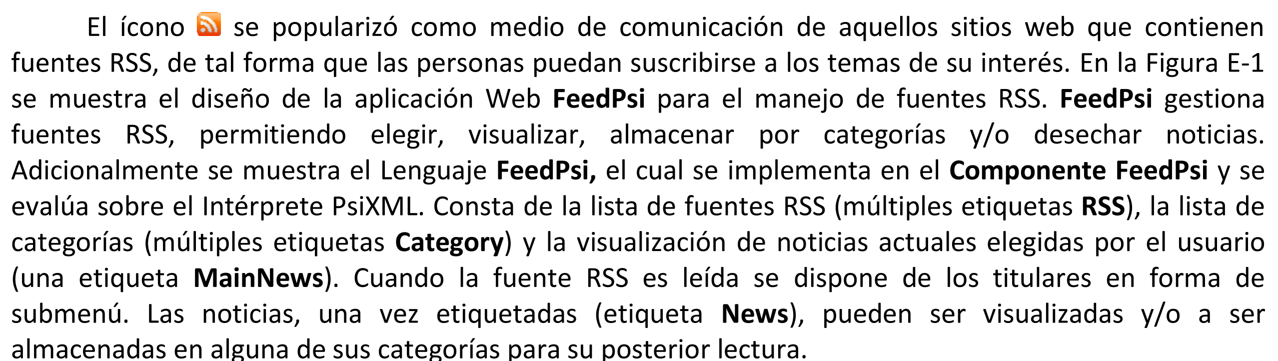
Anexo E. Casos de Estudio Psi

E.1. Aplicación Web FeedPsi

En el mundo de las noticias y la información, la difusión a través de diversos medios de comunicación es uno de los elementos claves para llegar a las personas. En el último siglo, los medios impresos, la radio y la televisión, han sido por excelencia la forma tradicional de estar al día de los acontecimientos. Con la llegada de Internet y los medios de comunicación social a través de ella, se ha revolucionado la forma de obtener y visualizar las noticias y la información. A través de la redifusión de contenidos web (de páginas web o blogs, denominadas **fuentes web**), los suscriptores reciben de manera actualizada la información. Con el uso de herramientas adecuadas (agregadores o lectores de fuentes web como Netvibes, Feedly, Feed Reader, Bloglines, etc.) los usuarios pueden mantener la información centralizada y clasificada.

Existen diferentes formatos para la gestión y visualización de las fuentes web, siendo los más utilizados RSS[13] (RSS Specifications, 2007) y Atom (IETF, 2012), ambos escritos en lenguaje XML. El objetivo de esta sección es ilustrar el uso de PsiEngine para la construcción y ejecución de una aplicación Web de difusión de noticias, a la que denominaremos **FeedPsi**.

- **Análisis y diseño**



El ícono se popularizó como medio de comunicación de aquellos sitios web que contienen fuentes RSS, de tal forma que las personas puedan suscribirse a los temas de su interés. En la Figura E-1 se muestra el diseño de la aplicación Web **FeedPsi** para el manejo de fuentes RSS. **FeedPsi** gestiona fuentes RSS, permitiendo elegir, visualizar, almacenar por categorías y/o desechar noticias. Adicionalmente se muestra el Lenguaje **FeedPsi,** el cual se implementa en el **Componente FeedPsi** y se evalúa sobre el Intérprete PsiXML. Consta de la lista de fuentes RSS (múltiples etiquetas **RSS**), la lista de categorías (múltiples etiquetas **Category**) y la visualización de noticias actuales elegidas por el usuario (una etiqueta **MainNews**). Cuando la fuente RSS es leída se dispone de los titulares en forma de submenú. Las noticias, una vez etiquetadas (etiqueta **News**), pueden ser visualizadas y/o a ser almacenadas en alguna de sus categorías para su posterior lectura.

La aplicación propuesta, tiene la ventaja de no requerir para su programación el uso de ninguna tecnología del servidor, siendo posible efectuar todo el despliegue de la aplicación en el cliente, con un mínimo coste computacional en el servidor.

[13] Forma abreviada para una fuente web en formato RSS.

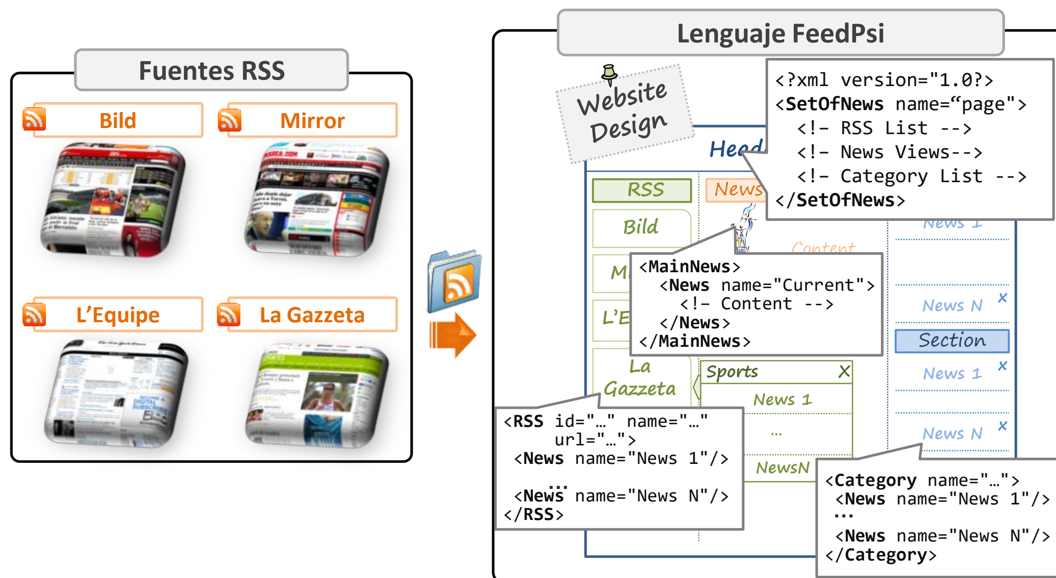
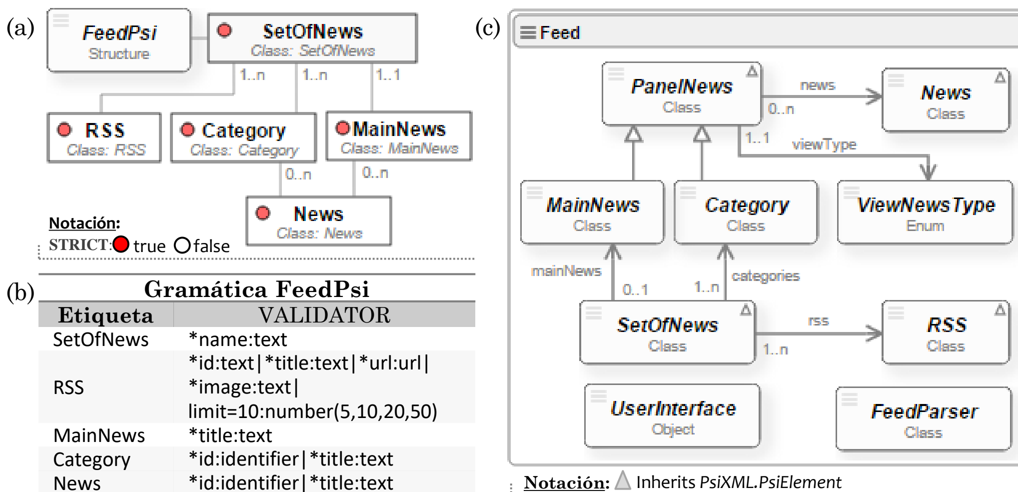


Figura E-1. Diseño de una aplicación Web para la visualización de noticas a través de fuentes RSS, mediante la creación de un Lenguaje Psi.

Componente Feed

La aplicación web para la gestión y visualización de fuentes RSS a la que acabamos de referirnos se construye mediante la especificación de un componente Psi, al que denominaremos **Componente Feed**. En Feed se implementará el lenguaje FeedPsi como se muestra en la Figura E-2, y tiene como objetivo crear de forma íntegra una aplicación web de una página. Para tal fin, se usará los pasos para la creación de lenguajes Psi de la sección 3.4.

En la Figura E-2(a) se muestra el PsiLSD del Lenguaje FeedPsi y en la Figura E-2(b) se muestra el PsiGVA para la gramática FeedPsi. La etiqueta raíz **SetOfNews** es el punto de partida para la evaluación de la aplicación *FeedPsi*. Contiene la lista de fuentes RSS (múltiples etiquetas **RSS**), la lista de categorías (múltiples etiquetas **Category**) y las noticias actuales (simple etiqueta **MainNews**).



Gramática FeedPsi

Etiqueta	VALIDATOR
SetOfNews	*name:text
RSS	*id:text|*title:text|*url:url|*image:text|limit=10:number(5,10,20,50)
MainNews	*title:text
Category	*id:identifier|*title:text
News	*id:identifier|*title:text

Figura E-2. (a) PsiLSD para el lenguaje FeedPsi. (b) PsiGVA para el lenguaje FeedPsi. (c) Diagrama de clases para el Componente Feed.

La etiqueta **RSS** tiene un identificador (atributo *id*), un nombre (atributo *name*) y la URL de la fuente RSS (atributo *url*). Tiene la capacidad de conectarse vía AJAX para obtener el fichero de noticias

del servicio RSS. La etiqueta **Category** posee la lista de noticias almacenadas por categoría (múltiples etiquetas News). Tiene un identificador (atributo *id*) y el título de la categoría (atributo *title*). La etiqueta **MainNews** contiene la lista de noticias que actualmente el usuario visualiza. Por último, la etiqueta **News** contiene la noticia obtenida de la fuente RSS y se le asigna un identificador único (atributo *id*) y se visualiza si está bajo la etiqueta **MainNews**.

En la Figura E-2(c) se muestra el diagrama de clases del Componente Feed. El detalle de implementación del Componente Feed se muestra en la URL: http://hilas.ii.uam.es/feedpsi/api en la pestaña "Classes Diagram", y su documentación respectiva en la pestaña "FeedPsiComponent". También se encuentra disponible en el CD Adjunto y en http://github.com/echavarriaga/FeedPsi.

En general, se implementa el conjunto de noticias (clase **SetOfNews**) donde se añaden y eliminan tanto fuentes RSS (mediante los métodos *addRSS*, *deleteRSS,* los cuales aparecen enumerados en la descripción de la clase) como categorías (métodos *addCategory*, *deleteCategory*. Cada fuente RSS (clase **RSS**) tiene la capacidad de actualizar su contenido (método *refresh*) realizando una llamada AJAX con la URL (atributo *url*), crear una noticia (método *createNews*), y crear un submenú de titulares con la ayuda de la clase **SetOfNews** (método *submenu*). Por su parte, una categoría (clase **Category**) y las noticias actuales (clase **MainNews**) puede añadir, eliminar o verificar existencia de noticias usando su clase base **PanelNews** (métodos *addNews*, *deleteNews*, *existNews*, respectivamente).

La clase **MainNews** se implementa para visualizar las noticias elegidas de las fuentes RSS o de las diferentes categorías. Por su parte, la clase **News** define una noticia y se le asigna un identificador único (atributo *id*). En el momento de crearse es llamado el método *_build*, el cual determina quién es su padre (atributo *parent* de la clase PsiElement) y pinta según el tipo de visualización de noticias (atributo *viewType*): si es "Complete" o "Titular", usando el objeto **UserInterface** para crear la interface noticia. También puede optarse por ser enviada a una categoría o viceversa (método *change*).

El Objeto **UserInterface** es el encargado de procesar y visualizar los fragmentos de interfaz de la aplicación FeedPsi (método *processInterface*). El método *init* inicializa todos los fragmentos de interfaz de la aplicación y los mantiene en memoria para su llamado.

- **Ejecutando la aplicación web FeedPsi**

Como ejemplo, se va a ilustrar la funcionalidad del componente FeedPsi junto con el lenguaje FeedPsi para resolver el problema de visualización de los servicios RSS de los periódicos: *Bild, Mirror, AS, La Gazzeta, L'Equipe y El Tiempo* (ver Figura 8-2), con la condición particular de no tener registradas noticias al inicio, y con las categorías de Football, Cicling, Tennis y Motor.

En el FRAGMENTO E-1 se muestra el código fuente Psi necesario para la ejecución de la aplicación web FeedPsi**.** Para la creación de la aplicación web se usó jQuery User Interface[14] (biblioteca de componentes visuales bajo la filosofía de jQuery).

FRAGMENTO E-1. Código fuente FeedPsi (sports.xml)[15].

```
<?xml version="1.0" encoding="utf-8"?>
<SetOfNews name="RSS Sports Psi Style">
  <RSS id="as" title="Germany: Bild Sport" image="images/rss/bild.png"
       url="http://rss.bild.de/bild-sport.xml"/>
  <RSS id="theguardian" title="England: The Guardian"
       image="images/rss/mirrorsport.png"
       url="http://feeds.theguardian.com/theguardian/uk/sport/rss"/>
  <RSS id="as" title="Spain: AS" image="images/rss/as.png"
       url="http://as.com/rss/tags/ultimas_noticias.xml"/>
  <RSS id="lagazzetta" title="Italy: La Gazzetta" image="images/rss/lagazzetta.png"
       url="http://www.gazzetta.it/rss/home.xml"/>
```

[14] http://www.jqueryui.com

[15] http://hilas.ii.uam.es/FeedPsi/source/sports.xml

```
  <RSS id="lequipe" title="France: L'Equipe" image="images/rss/lequipe.png"
       url="http://www.lequipe.fr/rss/actu_rss.xml"/>
  <RSS id="eltiempo" title="Colombia: El Tiempo" image="images/rss/eltiempo.png"
       url="http://www.eltiempo.com/deportes/rss.xml"/>
  <MainNews title="Latest Sports News"></MainNews>
  <Category id="football" title="Football"/>
  <Category id="cycling" title="Cycling"/>
  <Category id="tennis" title="Tennis"/>
  <Category id="motor" title="Motor"/>
</SetOfNews>
```

En la Figura 8-2 se muestra el resultado de la aplicación web FeedPsi, después de evaluar el programa “sport.xml” del fragmento anterior. En general, con la URL:

http://hilas.ii.uam.es/FeedPsi/index.html?list=<nombre>

se puede evaluar otros programas FeedPsi, donde <nombre> puede ser *spain*, *sports*, *sports-es* and *news*.

Finalmente cabe hacer notar que el objetivo final del caso de estudio presentado no es el de hacer una aplicación Web completamente depurada, sino mostrar cómo utilizar el concepto de componentes Psi, lenguajes Psi y el uso de PsiEngine para la evaluación de programas Psi, en este caso programas FeedPsi.

E.2. Animación Anisha

El objetivo de Anisha es construir dos XML-DSL para la animación de formas básicas de elementos SVG. El estudio de caso Anisha tiene como principal objetivo ilustrar el uso de multilenguaje Psi en el PsiEngine. En primer lugar, este caso de estudio permite validar la asociación de información heterogénea XML y JSON. En segundo lugar, prueba la coexistencia de varios lenguajes Psi y múltiples evaluaciones en el PsiEngine.

- **Componente Shapes**

El Lenguaje ShapesPsi tiene como objetivo la creación de círculos y rectángulos sobre un lienzo SVG. En la Figura E-3(a) se muestra el PsiLSD del Lenguaje ShapesPsi y en la Figura E-3(b) se muestra el PsiGVA para la gramática ShapesPsi. La etiqueta raíz **Shapes** es el punto de partida para la evaluación de un programa ShapesPsi. Esta etiqueta contiene círculos (múltiples etiquetas **Circle**) y rectángulos (múltiples etiquetas **Rectangle**).

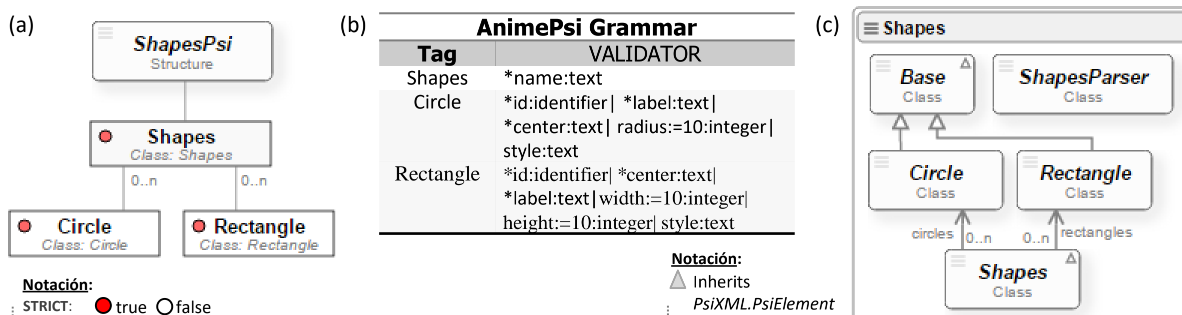


Figura E-3. (a) PsiLSD para el lenguaje ShapesPsi. (b) PsiGVA para el lenguaje ShapesPsi. (c) Diagrama de clases para el Componente Shapes.

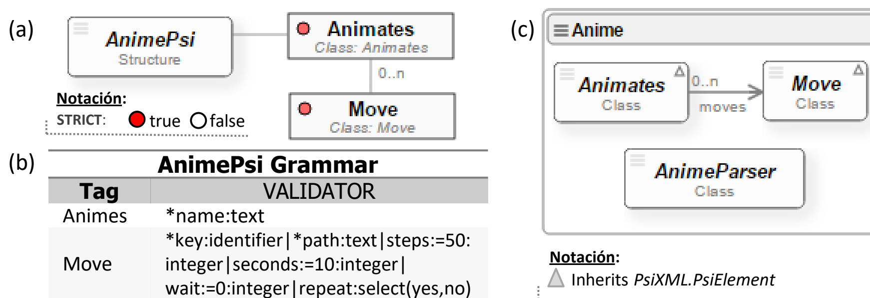
La etiqueta **Circle** define un círculo basado en la etiqueta **circle** de SVG. Contiene un identificador único (atributo *id*), el centro (atributo *center*), el radio (atributo *radius*), el rótulo (atributo *label*) y una

clase CSS (atributo *class*). Mientras que la etiqueta **Rectangle** define rectángulo basado en la etiqueta **rect** de SVG, contiene un identificador único (atributo *id*), el centro (atributo *center*), el ancho (atributo *width*), el alto (atributo *height*), el rótulo (atributo *label*) y una clase CSS (atributo *class*).

El diagrama de clases para el **Componente Shapes** se muestra en la Figura E-3(c). La clase *Shapes* define el conjunto de círculos y rectángulos. La clase abstracta *Base* define una figura genérica. La clase *Circle* pinta un círculo con un texto en el centro en un lienzo SVG, y la clase *Rectangle* pinta un rectángulo con un texto en el centro. Todos los círculos y rectángulos codificados se almacenan en PsiData del PsiEngine y pueden ser utilizados por otros programas y lenguajes Psi. La documentación detallada del Componente Shapes se encuentra en http://hilas.ii.uam.es/anisha/api. También se encuentra disponible en el CD Adjunto y en http://github.com/echavarriaga/Anisha.

■ Componente Anime

El Lenguaje AnimePsi tiene como objetivo la animación de Círculos y Rectángulos basados en el lenguaje ShapesPsi. En la Figura E-4(a) se muestra el PsiLSD del Lenguaje AnimePsi y en la Figura E-4(b) se muestra el PsiGVA para la gramática AnimePsi. La etiqueta raíz **Animes** es el punto de partida para la animación de figuras de la aplicación *Anisha*. Tiene la etiqueta **Move** para realizar el movimiento de las figuras. Está etiqueta define una referencia a un círculo o rectángulo de ShapesPsi (atributo *key*), el camino de desplazamiento basado en la etiqueta **path** de SVG (atributo *path*), la cantidad de pasos a realizar (atributo *steps*), el tiempo del movimiento (atributo *seconds*), el tiempo de espera antes de iniciar (atributo *wait*) y valor de repetición del movimiento (atributo *repeat*).



Tag	VALIDATOR
Animes	*name:text
Move	*key:identifier|*path:text|steps:=50:integer|seconds:=10:integer|wait:=0:integer|repeat:select(yes,no)

Figura E-4. (a) PsiLSD para el lenguaje AnimePsi. (b) PsiGVA para el lenguaje AnimePsi. (c) Diagrama de clases para el Componente Anime.

El **Componente Anime** implementa la funcionalidad asociada del lenguaje AnimePsi, como se muestra en el diagrama de clases de la Figura E-4(c). Este componente tiene dos clases: *Animates* y *Move*, asociados a las etiquetas **Animates** y **Move**, respectivamente. La clase *Animates* inicia el programa de animación, mientras que la clase *Move* implementa el movimiento de una figura. La documentación detallada del Componente Anime se encuentra en http://hilas.ii.uam.es/anisha/api.

■ Evaluación en Anisha

A continuación, como ejemplo, se presenta un juego sencillo basado en dos personajes con el fin de ilustrar el uso de los languages PsiLI y PsiCA de PsiEngine. En primer lugar, en el FRAGMENTO E-2 se muestra un fichero XML con información de los personajes.

FRAGMENTO E-2. Fichero de personajes *"characters.xml"*[16].

```
<?xml version="1.0" encoding="utf-8"?>
<Characters name="the characters ...">
  <Character id="p1" first="Luke" last="Skywalker"/>
  <Character id="p2" first="Obi-Wan" last="Kenobi"/>
```

[16] http://hilas.ii.uam.es/Anisha/code/characters.xml

```
</Characters>
```

En segundo lugar, en el FRAGMENTO E-3 se define un contexto escrito en JavaScript para el punto inicial y el color de los personajes cargados de “*characters.xml*” y registrados en PsiData.

FRAGMENTO E-3. Contexto en JavaScript para los personajes de “*characters.xml*”.

```
var context = {
  features: {
    p1: {point: "50,50", color: "black"},
    p2: {point: "250,200", color: "green"}
  }
}
PsiData.document.register("characters", "xml/characters.xml");
```

A continuación, en el FRAGMENTO E-4 se muestra un programa escrito en ShapesPsi para ilustrar los lenguajes PsiLI y PsiCA del PsiEngine. La etiqueta **Circle** define en el atributo *data-document* escrito en PsiLI cómo obtener la información del personaje cuyo identificador es “*p1*”, es decir, obtener el registro de “Luke Skawalker” (jQuery selector `characters:info=Character[id=p1]`), mientras que en el atributo *data-context* se obtiene el punto inicial y el color registrado en el *Contexto* (`features:qt=p1`). Con esta información asociada se actualizan los atributos *center*, *label* y *style* usando el PsiCA en la etiqueta **Circle**. De igual forma, se obtiene la información y se actualiza en la etiqueta **Rectangle** para el personaje “Obi-Wan Kenobi”.

FRAGMENTO E-4. Uso de los lenguajes PsiLI y PsiCA en los programas ShapesPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<Shapes name="the characters ...">
  <Circle data-document="characters:info=Character[id=p1]"
       data-context="features:qt=p1"
       id="cp1" center="{{qt.point}}" radius="30" label="{{info.first}}"
       style="stroke:black;stroke-width:1px;fill:{{qt.color}}"/>
  <Rectangle data-document="characters:info=Character[id=p2]"
       data-context="features:qt=p2"
       id="cp2" center="{{qt.point}}" width="60" height="40" label="{{info.first}}"
       style="stroke:black;stroke-width:1px;fill:{{qt.color}}"/>
</Shapes>
```

En el FRAGMENTO E-5 se define un programa escrito en lenguaje AnimePsi. En la evaluación de este programa “Luke Skawalker” se acerca a “Obi-Wan Kenobi” y este sale corriendo. La animación se puede ver en http://hilas.ii.uam.es/anisha/runs.

FRAGMENTO E-5. Juego de movimiento de los personajes usando el lenguaje AnimePsi.

```
<?xml version="1.0" encoding="utf-8"?>
<Animates name="runs, Obi-Wan, runs">
  <Move key="cp1" steps="50" seconds="3" path="L100,50L180,200"/>
  <Move key=" key="cp2" steps="50" seconds="5" wait="4"
        path="C250,100 400,100 400,200 S550,300 550,200 C550,50 100,50 100,200 L100,300 L250,300
l250,200" />
</Animates>
```

E.3. Diagramación Grapher

El objetivo de este caso de estudio es ilustrar la creación de diagramas donde su apariencia y contenido dependa de múltiples fuentes de información XML y JSON. En este anexo se detallan los aspectos generales para la creación y visualización de diagramas, se resumen los detalles de implementación de los componentes Graphs y Paint, y se ilustra con un ejemplo el uso de estos lenguajes. Por otro lado, para facilitar la creación de diagramas a partir de estos componentes, se creó la aplicación web

denominada **Grapher**. Esta aplicación es un entorno de desarrollo ligero basado en proyectos y en la edición de ficheros XML.

- **Análisis y diseño**

En la Figura E-5 se muestran las partes que intervienen para la creación y visualización de un diagrama: la *librería gráfica*, las *fuentes de información* y la *diagramación*. La *librería gráfica* tiene como objetivo definir elementos gráficos basados en elementos Scalable Vector Graphics (SVG) (W3C, 2011), que sean manipulables, transformables y reutilizables. Las *fuentes de información* son los ficheros de datos XML o JSON asociados al diagrama y median en su apariencia y contenido. Por último, la *diagramación* es la representación gráfica del diagrama con elementos SVG a partir de la *librería gráfica* y las *fuentes de información*. Esta representación se hace directamente sobre un lienzo SVG (elemento **svg** de SVG).

La creación de un diagrama se desarrolla a partir del concepto de **definición gráfica**. Para el marco de trabajo del caso de estudio, una definición gráfica es la abstracción del diagrama a partir de dos elementos gráficos: **figura** y **línea**. Un elemento gráfico define una plantilla o esqueleto de elementos SVG. Es decir, la **figura** es la agrupación de elementos SVG tales como: rect, circle, ellipse, line, polyline, polygon, text, path, etc. Mientras que una **línea** es un elemento SVG: line, polyline o path. Además, a un elemento gráfico se le pueden especificar cuáles son sus elementos SVG modificables (modificación de texto, de imágenes y de estilo) y cuáles son sus elementos SVG que intervienen en su comportamiento (arrastre, visualización, tamaño y rotación), como se muestra en la Figura E-5.

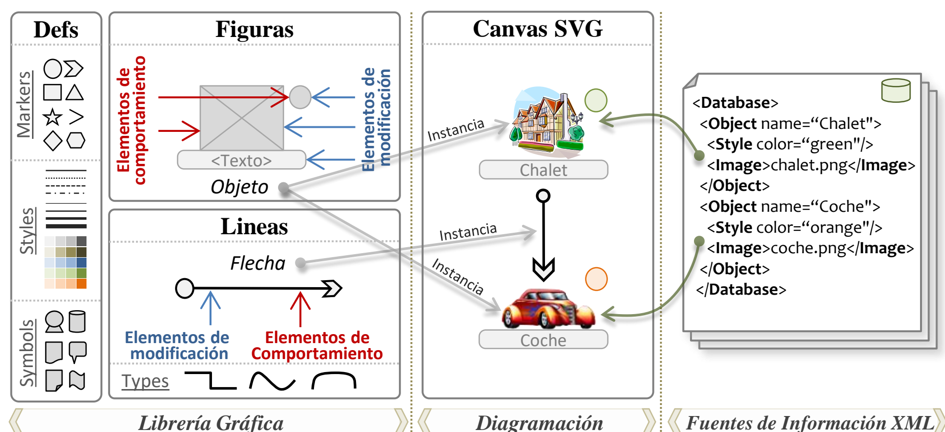


Figura E-5. Creación y visualización de figuras y líneas basadas en el concepto de definición gráfica.

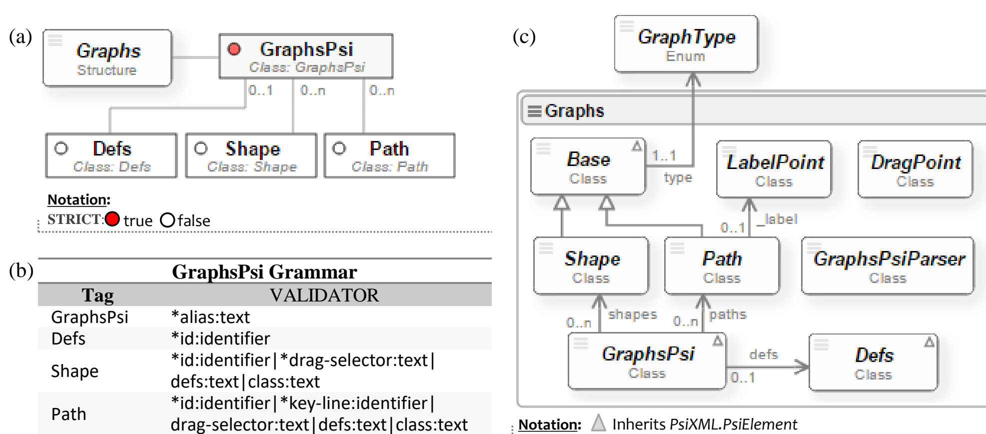
En la Figura E-5 el elemento *Objeto* está conformada por: una imagen, un texto y un círculo. Cuando se crea una instancia del elemento *Objeto* se modifica la imagen, el texto y se da un estilo de círculo (cambiar color de relleno y color de contorno). Esta modificación puede provenir de procesar la información externa al diagrama, es decir, de procesar fuentes de información XML (o JSON). En este entorno, "instanciar" un elemento gráfico es clonar todos los elementos SVG de la librería y dar una ubicación visible dentro de un lienzo SVG. Por otra parte, en la Figura E-5 el elemento *Flecha* está conformado por: el tipo de línea y los marcadores inicial y final (obtenidos de Defs). Al igual que el elemento *Objeto*, estos elementos pueden ser modificados por las fuentes de información XML (o JSON).

En consecuencia, para crear y visualizar diagramas asociando fuentes de información, se plantean los siguientes componentes Psi: Componente **Graphs** y Componente **Paint**. En las próximas secciones se resumirá la implementación de estás componentes.

■ **Componente Graphs**

El **Lenguaje GraphsPsi** administra los elementos gráficos de la *librería gráfica* definiendo las siguientes etiquetas: **GraphsPsi**, **Defs**, **Shape** y **Path**. En la Figura E-6(a) se muestra el diagrama de estructura del lenguaje GraphsPsi, y en la Figura E-6(b) se muestra la validación de atributos de la gramática GraphsPsi. Según Figura E-6(a), la etiqueta raíz **GraphsPsi** es el punto de partida para la definición de la *librería gráfica*. Tiene un alias (atributo *alias*) y es opcional. Además, contiene las siguientes etiquetas:

- Etiqueta **Defs** (opcional): contiene las definiciones de SVG como símbolos, marcas, estilos, etc. Tiene un identificador (atributo *id*) para cada definición SVG para ser referenciados por los elementos gráficos.
- Etiqueta **Shape** (opcional, múltiple): define una figura mediante un conjunto de elementos gráficos SVG (rect, circle, ellipse, line, polyline, polygon, text, path, etc.). Tiene un identificador (atributo *id*) y un selector de arrastre (atributo *drag-selector*) que delega a uno o más elementos gráficos el movimiento de la figura. Además, el atributo *defs* lista las definiciones SVG y el atributo *class* se usa para dar estilo CSS que la figura requiera.
- Etiqueta **Path** (opcional, múltiple): define una línea con una de las etiquetas SVG: line, polyline o path. Tiene un identificador (atributo *id*) y un selector de arrastre (atributo *drag-selector*) para dar movimiento a la línea. Los atributos *defs* y *class* funcionan de manera análoga que en la etiqueta **Shape**.



GraphsPsi Grammar

Tag	VALIDATOR
GraphsPsi	*alias:text
Defs	*id:identifier
Shape	*id:identifier|*drag-selector:text|defs:text|class:text
Path	*id:identifier|*key-line:identifier|drag-selector:text|defs:text|class:text

Figura E-6. (a) PsiLSD para el lenguaje GraphsPsi. (b) PsiGVA para la gramática GraphsPsi. (c) Diagrama de clases del componente Graphs.

En el FRAGMENTO E-6 se muestra la plantilla de definición para cada uno de los elementos del Lenguaje GraphsPsi.

FRAGMENTO E-6. Plantilla de definición del Lenguaje GraphsPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<GraphsPsi alias="sample">
  <Defs id="…">
    <svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
      <defs> <!-- Contenido de definiciones SVG --> </defs>
    </svg>
  </Defs>
  <Shape id="…" drag-selector="…" defs="key1,key2,…">
    <svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
      <!-- Elementos gráficos SVG -->
    </svg>
  </Shape> ...
```

```
  <Path id="…" key-line="…" class="pert" defs="key1,key2,…" drag-selector="…">
    <svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
      <!-- Elementos gráficos SVG -->
    </svg>
  </Path> ...
</GraphsPsi>
```

El **Componente Graphs** implementa la funcionalidad asociada al lenguaje GraphsPsi y en la Figura E-6(c) se muestra su diagrama de clases. Por facilidad, la clase asociada a cada etiqueta tiene el mismo nombre, por ejemplo, la etiqueta **Shape** tiene la clase *Shape*, y así respectivamente. Cabe mencionar que las clases *GraphsPsi*, *Base* y *Defs* heredan de la clase abstracta *PsiElement* del PsiXML. El Componente Graphs, cuando se inicializa crea tres clasificadores en **PsiData** del *EnginePsi*: “shape”, “path” y “defs”. En cada uno de ellos se almacenará y se pondrá a disposición las figuras, las líneas y las definiciones SVG, respectivamente.

Según la Figura E-6(c), la clase *GraphsPsi* administra las definiciones SVG, la lista de líneas y la lista de figuras. La clase *Defs* registra todas sus definiciones SVG en “defs” de PsiData con su identificador. La clase *Base*, administra el documento SVG del elemento gráfico (figura o línea) y registra su instancia según corresponda en PsiData. Crea la funcionalidad de arrastre con la ayuda del atributo *drag-selector* y le asocia el estilo de clases CSS con el atributo *class*. Además, tiene la capacidad de clonar los elementos SVG del documento para “instanciar” el elemento gráfico. Las clases *Shape* y *Path* heredan de la clase *Base* y tiene como objetivo personalizar la funcionalidad de arrastre, escala y rotación. La clase *DragPoint* crea un punto dinámicamente para el movimiento de los extremos de una línea, mientras que la clase *LabelPoint* crea un rótulo en la línea.

El diagrama de clases detallado del componente Graphs se encuentra en http://hilas.ii.uam.es/grapher/api, en la pestaña “*Class Diagram*”. La documentación de los elementos de programación se encuentra en la pestaña “*Graphs Component*”. También se encuentra disponible en el CD Adjunto y en http://github.com/echavarriaga/Grapher.

■ **Componente Paint**

El Lenguaje **PaintPsi** tiene como objetivo la *diagramación* de figuras y líneas basadas en el lenguaje GraphsPsi, para ello define las siguientes etiquetas: **PaintPsi**, **Include**, **Script**, **Layer**¸ **Figure**, **Line**, **Setting** y **Modify**.

En la Figura E-7(a) se muestra el diagrama de estructura del lenguaje PaintPsi y en la Figura E-7(b) se muestra la validación de atributos de la gramática PaintPsi.

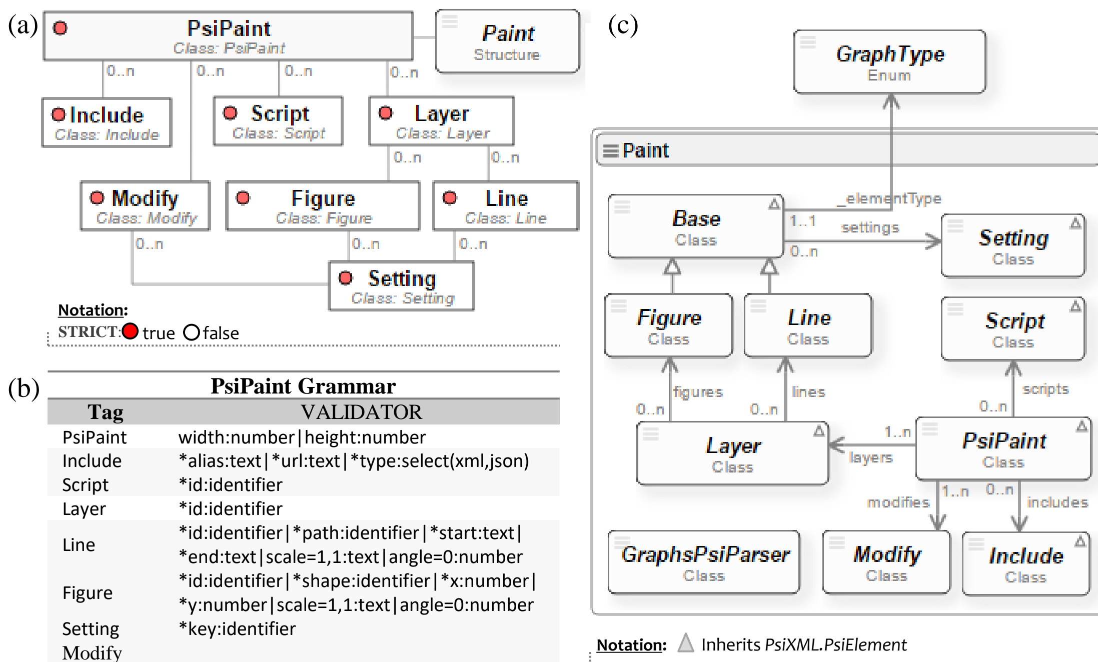
Según la Figura E-7(a) la etiqueta raíz **PaintPsi** especifica el ancho (atributo *width*) y el alto (atributo *height*) del lienzo SVG activo. Además, contiene las etiquetas:

- Etiqueta **Include** (opcional, múltiple): adiciona información XML o JSON con un alias (atributo *alias*) en PsiData (en los clasificadores “document” y “json”, respectivamente) del PsiXML.
- Etiqueta **Script** (opcional, múltiple): define un script que se ejecuta cuando se evalúa el programa Psi.
- Etiqueta **Layer** (opcional, múltiple): define el concepto *capa gráfica* donde contiene figuras, líneas y scripts. El concepto de capa gráfica hace referencia a la profundidad en la que está el grupo de elementos de la capa, por ejemplo, si está al fondo (se define como primera etiqueta) o al frente (se define como última etiqueta). A continuación listaremos sus etiquetas:
  - Etiqueta **Figure** (opcional, múltiple): instancia una figura basada en un elemento de la *librería gráfica*. Para ello dispone de un identificador (atributo *id*), la referencia a la instancia *Shape* (atributo *shape*), las coordenadas de ubicación x (atributo *x*) e y (atributo *y*), la escala (atributo *scale*) y el ángulo de inclinación (atributo *angle*).
  - Etiqueta **Line** (opcional, múltiple): para esta versión del lenguaje, se instancia una línea recta basada en una línea (etiqueta line de SVG) de la *librería gráfica*. Dispone un identificador

(atributo *id*), referencia a la instancia *Path* (atributo *path*), la coordenada del punto inicial (atributo *start*), la coordenada del punto final (atributo *end*), la escala (atributo *scale*) y el ángulo de inclinación (atributo *angle*).

- Etiqueta **Modify**: modifica un elemento gráfico (**Line** o **Figure**) previamente creado mediante el uso de etiquetas **Setting**.

La etiqueta **Setting** (opcional, múltiple) define y modifica elementos SVG tanto de una figura como de una línea. Tiene una clave (atributo *key*) que hace referencia al elemento SVG para aplicar el cambio.



PsiPaint Grammar

Tag	VALIDATOR
PsiPaint	width:number|height:number
Include	*alias:text|*url:text|*type:select(xml,json)
Script	*id:identifier
Layer	*id:identifier
Line	*id:identifier|*path:identifier|*start:text|*end:text|scale=1,1:text|angle=0:number
Figure	*id:identifier|*shape:identifier|*x:number|*y:number|scale=1,1:text|angle=0:number
Setting	*key:identifier
Modify	

Figura E-7. (a) PsiLSD para el lenguaje PaintPsi. (b) PsiGVA para la gramática PaintPsi. (c) Diagrama de clases del componente Paint.

En el FRAGMENTO E-7 se muestra la plantilla de definición para cada uno de los elementos del lenguaje PaintPsi.

FRAGMENTO E-7. Plantilla de definición del Lenguaje PaintPsi.

```
<?xml version="1.0" encoding="utf-8"?>
<PaintPsi xmlns:xlink="http://www.w3.org/1999/xlink" width="…" height="…">
  <Include alias="…" url="…" type="…"/> ...
  <Layer id="…" class="…">
    <Figure id="…" x="x" y="y" shape="…" data-document="…" scale="…" angle="…">
      <Setting key="…" …="…" …/> ...
    </Figure> ...
    <Line id="…" path="…" start="x1,y1" end="x2,y2" scale="…" angle="…">
      <Setting key="…" …="…" …="…"/> ...
    </Line> ...
    <Script id="…"><!-- Definición de Scripts --></Script> ...
  </Layer> ...
  <Script id="…"><!-- Definición de Scripts --></Script> ...
</PaintPsi>
```

El **Componente Paint** implementa la funcionalidad asociada al lenguaje PaintPsi, como se muestra en el diagrama de clases de la Figura E-7(c). Al igual que en el componente Graphs, la clase asociada a cada etiqueta tiene el mismo nombre, es decir, la etiqueta **PaintPsi** tiene la clase *PaintPsi*, la etiqueta **Include** tiene la clase *Include*, y así sucesivamente.

La clase raíz *PaintPsi* administra las capas gráficas, las inclusiones y los scripts. Inicializa el contexto y adiciona todas las capas que tenga a disposición en el lienzo SVG activo. La clase *Include* es la encargada de traer un documento XML o un objeto JSON y registrarlo en PsiData. Mientras que la clase *Script* registra un script en PsiData con un identificador (atributo *id*). El script se usa para agregar dinamismo al diagrama y al evaluarse, tiene a su disposición como variables de entorno: el lienzo SVG activo, el PsiData, y los elementos gráficos actuales del diagrama.

La clase *Layer* tiene como objetivo crear y administrar una *capa gráfica*. Usa como agrupador la etiqueta **g** de SVG y lo adiciona al lienzo SVG activo con un identificador (atributo *id*) y una clase CSS (atributo *class*). Además, pinta su la lista de figuras y la lista de líneas.

La clase *Base* implementa el proceso de pintado tanto de figuras como de líneas. Para ello, usa la referencia de la definición gráfica (atributo *graph* o *path*, respectivamente) y crea una instancia (figura o línea, respectivamente). Además, la clase *Base* tiene la funcionalidad de modificar el tamaño, de aplicar la rotación, y de realizar los cambios de configuración de elementos SVG de un elemento gráfico a través de las instancias de *Setting*. La clase *Setting* busca el elemento SVG con el atributo *key* y efectúa los cambios de configuración.

La clase *Figure* hereda de la clase *Base* y tiene como objetivo pintar una figura. Tiene la funcionalidad de mover la figura a una nueva posición y de informar sobre su tamaño (ancho y alto). La clase *Line* hereda de la clase *Base* y tiene como objetivo pintar una línea recta (alcance actual del lenguaje). Tiene la capacidad de mover el punto inicial y el punto final. Para implementar una polilínea o líneas curvas de Bessel es necesario tipificar la clase *Line* y adicionar el comportamiento propio de cada una de ellas.

Al igual que el componente Graphs, el diagrama de clases detallado y la documentación del componente Paint se encuentra en http://hilas.ii.uam.es/grapher/api. También se encuentra disponible en el CD Adjunto.

- **Ejemplo ilustrativo**

En la Figura E-8 se muestra un fragmento de código de la *librería gráfica* en lenguaje GraphsPsi (fichero "library.xml") y un fragmento de código de la *diagramación* en lenguaje PaintPsi (fichero "maths.xml"). Este ejemplo resalta el encadenamiento de los lenguajes GraphsPsi y PaintPsi, con la información XML (fichero "db.xml"). El ejemplo está disponible en la aplicación Grapher, en el proyecto "Sample".

El fragmento "library.xml" pone a disposición los siguientes elementos: Defs (Symbols id=*"home"* e id=*"user"*; Markers id=*"circle"*; Styles id=*"styles"*), Shapes (id=*"element"*) y Paths (id=*"connection"*).

El fragmento "maths.xml" carga el fichero XML (con url="files/Sample/data/db.xml") y queda disponible con el alias *db*. Se puede apreciar que la **Figure** con *id*="p1" es desplazada a la posición (250,50) y tiene la referencia al elemento gráfico **Shape** con *id*="element". Además, con el uso del lenguaje PsiLI se asocia el node **Person** con *id*="1" (atributo *data-document*) del fichero "db.xml". Para obtener el nombre "Évariste Galöis" se usa el lenguaje PsiCA y se extrae del fichero XML a través de la etiqueta **Setting** (*text*="{{p.name}}"), este nombre es adicionado como texto en la etiqueta **text** de SVG. El símbolo usado para la figura *id*="p1" es el "user" (*href*="#user"). El resultado visual de este encadenamiento se ve en la Figura E-8 en el Canvas SVG Activo.

- **Aplicación Web Grapher**

Una vez implementados los componentes reutilizables Graphs y Paint, en la Figura E-8 se muestra la implementación de la aplicación web denominada **Grapher**. Está aplicación es un entorno de desarrollo ligero para la creación de diagramas y modelos gráficos basados en proyectos, en la edición de ficheros XML y en el uso de los componentes Graphs y Paint. Grapher se divide en dos secciones:

Proyecto y *Diagrama Psi*. Un *proyecto* es un conjunto de ficheros de librerías gráficas (ficheros GraphsPsi), diagramación (ficheros PaintPsi) y fuentes de información XML (ficheros XML). El *Diagrama Psi* es un canvas SVG donde se evaluarán los programas de diagramación. A continuación se describen las principales funcionalidades de Grapher:

— **Project Menu**: crea, abre o cierra un *proyecto*. Crear un *proyecto* implica crear en el servidor un directorio con el *alias* del *proyecto*, y en él se crean tres subdirectorios: *graphs* (aloja ficheros de programas GraphsPsi), *paints* (aloja ficheros de programas PaintPsi) y *data* (aloja ficheros de datos XML y JSON). Abrir un *proyecto* es traer la estructura del directorio y administrarla en la sección del *Proyecto*, cada subdirectorio se muestra en su pestaña correspondiente (Graphs Psi, Paint Psi y Data). Eliminar un *proyecto*, implica eliminar el directorio del servidor. Además, hay herramientas de depuración, limpiar el diagrama, ver el fichero SVG generado y mostrar el código fuente PaintPsi.

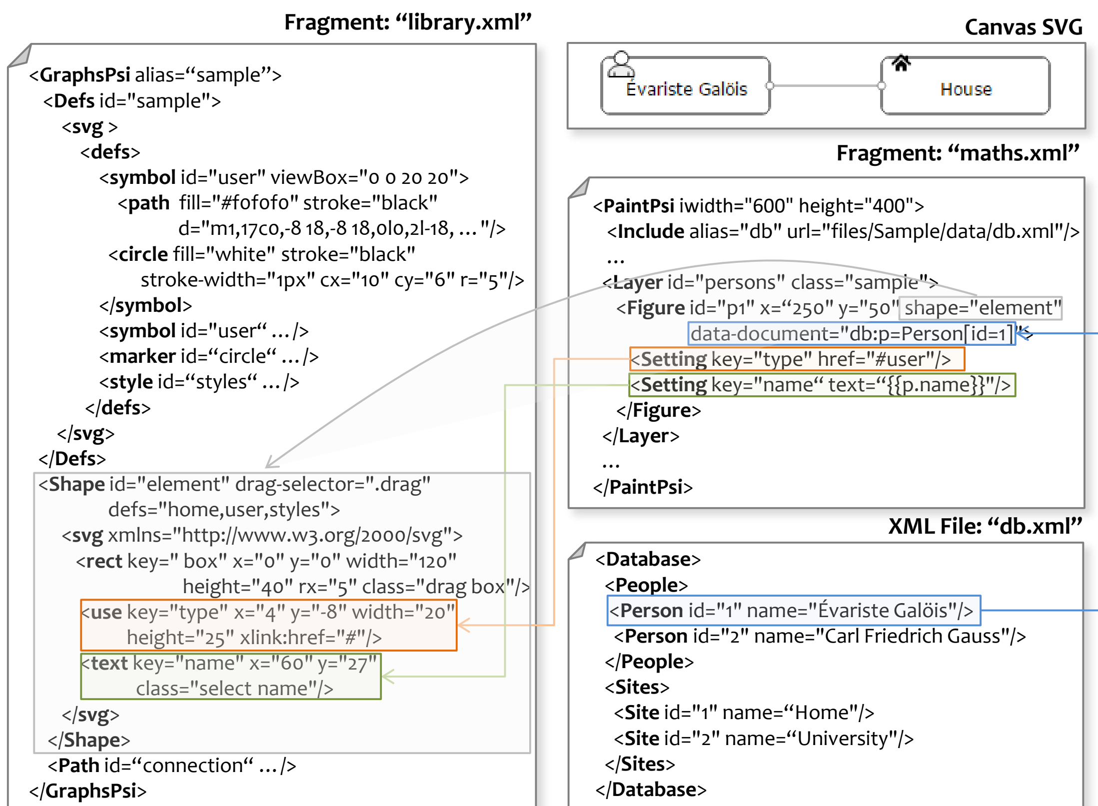


Figura E-8. Ejemplo ilustrativo del encadenamiento de los lenguajes GraphsPsi y PaintPsi, con asociación de información XML.

— **GraphsPsi Editor**: edita un fichero de código fuente GraphsPsi del proyecto activo. El editor proporciona herramientas de autocompletar para la gramática GraphsPsi. Al evaluar un fichero de GraphsPsi, los elementos gráficos quedan disponibles en PsiData del *PsiEngine*, y aumenta en 1 su ejecución y se resalta en verde.

— **PaintPsi Editor**: edita un fichero de código fuente PaintPsi del proyecto activo. Al igual que el Editor GraphsPsi, tiene autocompletar para la gramática PaintPsi. Al evaluar un fichero PaintPsi, su diagramación se construye sobre el canvas SVG de la sección *Diagrama Psi*, y aumenta en 1 su ejecución y se resalta en verde. Cabe notar que si se evalúa dos veces un mismo programa, se modifica los elementos gráficos más no se adicionan nuevos.

— **XML Editor**: edita ficheros de información XML.