Computer Languages, Systems and Structures 53 (2018): 90 – 120

DOI: https://doi.org/10.1016/j.cl.2018.02.002

Automated modelling assistance by integrating heterogeneous information sources

Ángel Mora Segura[a], Juan de Lara[a], Patrick Neubauer[b], Manuel Wimmer[b]

[a] *Modelling & Software Engineering Research Group*
http://miso.es
Universidad Autónoma de Madrid (Spain)
[b] *CDL-MINT, Business Informatics Group*
https://www.big.tuwien.ac.at
TU Wien (Austria)

Abstract

Model-Driven Engineering (MDE) uses models as its main assets in the software development process. The structure of a model is described through a meta-model. Even though modelling and meta-modelling are recurrent activities in MDE and a vast amount of MDE tools exist nowadays, they are tasks typically performed in an unassisted way. Usually, these tools cannot extract useful knowledge available in heterogeneous information sources like XML, RDF, CSV or other models and meta-models.

We propose an approach to provide modelling and meta-modelling assistance. The approach gathers heterogeneous information sources in various technological spaces, and represents them uniformly in a common data model. This enables their uniform querying, by means of an extensible mechanism, which can make use of services, e.g., for synonym search and word sense analysis. The query results can then be easily incorporated into the (meta-)model being built. The approach has been realized in the Extremo tool, developed as an Eclipse plugin.

Extremo has been validated in the context of two domains – production systems and process modelling – taking into account a large and complex industrial standard for classification and product description. Further validation results indicate that the integration of Extremo in various modelling environments can be achieved with low effort, and that the tool is able to handle information from most existing technological spaces.

Keywords: Modelling, (Meta-)modelling, Modelling assistance, Domain-specific languages, Language engineering

Email addresses: Angel.MoraS@uam.es (Ángel Mora Segura), Juan.deLara@uam.es (Juan de Lara), neubauer@big.tuwien.ac.at (Patrick Neubauer), wimmer@big.tuwien.ac.at (Manuel Wimmer)

1. Introduction

Model-Driven Engineering (MDE) advocates an active use of models throughout the software development life-cycle. Thus, models can be used to specify, analyse, test, simulate, execute, generate code and maintain the software to be built, among other activities [1, 2, 3].

Models are sometimes built with general-purpose modelling languages, such as the Unified Modelling Language (UML) [4]. In other cases, modelling is performed using Domain-Specific Languages (DSLs) [5]. DSLs contain tailored domain-specific primitives and concepts accurately representing the abstractions within a domain, which may lead to simpler, more intensional models. The abstract syntax of a DSL is described by a meta-model, which is itself a model. Meta-models are typically built using class diagrams, describing the set of models considered valid. Thus, the construction of models and meta-models is a recurrent and central activity in MDE projects [6].

High quality models and meta-models are pivotal for the success of MDE projects. They capture the most important concepts of a domain or describe the features of a system. Nevertheless, they are mostly built in an unassisted way, with no mechanisms for reusing existing knowledge. This situation contrasts with modern *programming* IDEs, which support code completion or provide help for using a given API [7, 8]. However, in the MDE field, the modeller normally has the burden of creating the model from scratch. For this reason, modellers would greatly benefit from flexible access and reuse of existing knowledge in a domain. This knowledge might be stored on various technological spaces [9, 10], including the modelling technical space, but also the XML, ontologies, and RDF technical spaces.

In order to improve this situation, we propose an extensible approach that provides assistance during the modelling process. In our proposal, we extract the information from an extensible set of different technical spaces. For example, in the XML technical space, DTDs or XML schemas as well as specific XML documents are covered by the assistant; while in the modelling technical space, meta-models and models can be queried. This heterogeneous information is stored in a common data model, so that it can be queried and visualized in a uniform way. The query mechanism is extensible and can make use of services, e.g. for synonym search or word sense analysis. The results of the queries are prioritized and aggregated for all information sources in the repositories and can then be incorporated into the (meta-)model under construction.

We have realized this concept in Extremo and provide an open source Eclipse plugin, which is freely available at the Extremo project website[1]. The web site includes short videos illustrating the main concepts explained in this paper as well as a set of resources, which have been used during the evaluation. Extremo's architecture is extensible and modular by the use of Eclipse extension points, and enables the addition of new information sources and types of

[1] `http://miso.es/tools/extremo.html`

queries. The assistant has been designed to be easily integrated with external modelling environments, also through extension points.

We have evaluated our approach under several perspectives. First, we show EXTREMO's usefulness to create DSLs in two case studies. The first one is in the area of process modelling for immigration procedures and the second is in the area of standard-conforming industrial production systems. We have evaluated its extensibility by describing its integration with a variety of modelling tools, ranging from graphical to tree-based editors. In order to evaluate format extensibility (i.e., the ability to import from new technical spaces), we perform an analytical evaluation of the degree of coverage of the data model. The query mechanism is tested by describing a catalogue of common queries for object-oriented notations. Finally, we address a discussion and the lessons learned from the results of the evaluation.

In comparison with our previous work [11], we provide extensions for a set of different technical spaces that include constraint interpreters and an extensible query mechanism. Moreover, EXTREMO's internal data model has been extended to handle level-agnostic information, i.e., for an arbitrary number of meta-levels. For example, we have integrated XML schemas and multi-level models [12] as information sources, and integrated EXTREMO with further modelling and meta-modelling environments. Finally, we report on an evaluation based on process modelling and production systems domain case study, using the eCl@ss standard [13] and provide an analytical evaluation on the generality of the data model we propose.

The rest of this paper is organized as follows. Section 2 provides an overview of the approach and its motivation. Section 3 explains the main parts of the assistant: the handling of heterogeneous sources (Section 3.1), the ability to perform queries on them in a uniform and extensible way (Section 3.2), and the handling of constraints (Section 3.3). Section 4 describes the extensible and modular architecture of the assistant, and how it can be integrated with modelling and meta-modelling tools. Section 5 evaluates the approach under three different perspectives, which include the usefulness for (*i*) language engineering, (*ii*) data extensibility, and (*iii*) integrability with external tools. Section 6 compares with related work, and Section 7 presents the conclusions and lines for future research.

2. Motivation and overview

Many technical tasks in software engineering require from access to knowledge found in a variety of formats, ranging from documents in natural language, to semi-structured and structured data. There is a current trend to make such information readily available and easy to embed in different types of artefacts generated during the software construction process [14]. For example, in the programming community, there are efforts to profit from code repositories, and Q&A sites like StackOverflow to automate coding and documentation tasks [15, 16, 17]. Some of these approaches are based on a phase of artefact collection, followed by their preprocessing and storage into a uniform database,

which then can be queried using appropriate languages [16]. Following this trend, our objective is to make available to the model engineer a plethora of (possibly heterogeneous) resources that can be queried in a uniform way, and embedded into the model being built.

In general, the task of creating a high quality meta-model is complex because it involves two roles: (*i*) a domain expert, who has in-depth knowledge of a particular domain and (*ii*), a meta-modelling expert, who is experienced in object-oriented design and class-based modelling. Nevertheless, many times, the meta-modelling expert is left alone in the construction of a meta-model, or needs to make a decision based on tacit domain knowledge or under-specified language requirements. In this scenario, the meta-modelling expert takes the role of the domain expert too, which may lead to mistakes or omissions, compromising the quality of the meta-model.

Meta-models within a domain are not completely different from each other, but they sometimes have recurring patterns and use common idioms to represent concepts [18, 19]. For example, while building a language to describe behaviour, designers normally resort to accepted specification styles, including variants of languages such as state machines, workflow, rule-based or data-flow languages, enriched with domain-specific elements. The language designer can obtain this information from sources like meta-models, class diagrams, ontologies, XML schema definitions (XSD) or RDF documents. Moreover, having access to a variety of information sources helps in obtaining the necessary domain knowledge, vocabulary and technical terms required to build the meta-model. This situation also applies when building models, instances of a given meta-model. In this case, it may be helpful to have a way to query information sources and knowledge bases. These queries may use the data types present in the meta-model to help filtering the relevant information.

For this purpose, we have devised a modelling assistance approach, whose working scheme is shown in Figure 1. The approach is useful for creating models at any meta-level. Our proposal is based on the creation of a set of repositories (label 1 in the Figure), in which heterogeneous data descriptions (OWL ontologies, Ecore meta-models, RDF Schemas, XML schema definitions), and data sources (OWL, RDF data, EMF models, XML documents) are injected.

Our system represents this heterogeneous data using a common data model, so that information sources can be stored in the repository in a uniform way. The system provides extensible facilities for the uniform and flexible query of the repository (label 2). We provide basic services for synonym search and word sense analysis, and a predefined catalogue of queries, which can be externally extended. The repository can also store heterogeneous constraints, and we support their evaluation using an extensible facility (label 3). The results of the queries for each source in the repository are aggregated and ranked according to their suitability and relevance. These query results and the information sources themselves can be visualized (label 4). Although the assistance system is independent of any modelling tool, it has been designed to be easy to integrate with a wide range of tools (label 5).

We have identified several scenarios where the assistant is useful. They can

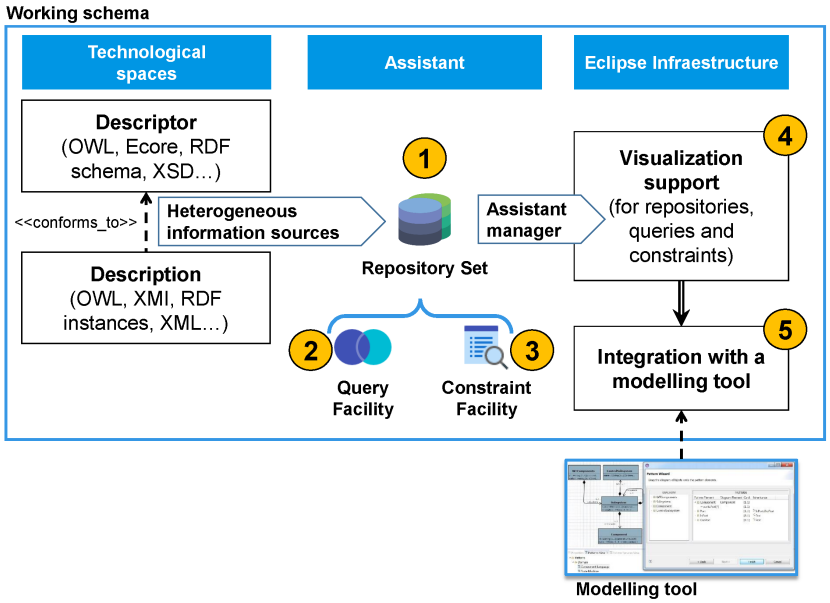

Figure 1: Overview of our approach

be generally classified in three areas. Firstly, for creating models and meta- models. Second, to create artefacts describing a set of other artefacts, like in model-based product lines [20, 21] or model transformation reuse [22]. Finally, to evaluate quality aspects of a set of (perhaps heterogeneous) resources. More in detail, our approach is useful:

- As support for the development of new domain meta-models. This way, domain concepts and vocabulary can be sought in external sources such as XML documents or ontologies.
- To create models for a particular domain. In this case, model elements conforming to the same or similar meta-model can be incorporated into the model being built, and heterogeneous information sources can be queried in order to extract concrete data values.
- To design a "concept" meta-model [22]. A concept is a minimal meta-model that gathers the core primitives within a domain, e.g., for workflow languages. Furthermore, concepts can be used as the source meta-model of a model transformation, becoming reusable, so they can be bound to a particular meta-model. This task implies the querying and understanding of a variety of meta-models for a particular domain and therefore the assistant becomes useful.
- To aggregate multiple existing models into a model-based product line [20, 21]. In this approach, a description of the space of possible features of a

software system is created, typically through a feature model. The choice of features implies the selection of a software model variant. This way, there is a need for understanding an existing family of models and to describe their variability. One possible approach is to merge or superimpose all model variants (leading to a so called 150% model) and use "negative variability", which selectively removes deselected artefacts [20, 21].

- To detect "bad smells" [23] or signs of bad modelling practices [24] in a set of resources. This is possible, as we can perform queries over a repository to e.g., detect isolated nodes, or find abstract classes with no children. Moreover, our query mechanism is extensible, so that technology-specific or domain-specific queries can be created.

While our approach is useful in all these scenarios, to focus the paper, we concentrate on the modelling and meta-modelling scenarios.

3. The ingredients of a (meta-)modelling assistant

In this section, we detail the three main parts of our approach: (*i*) the handling of heterogeneous sources through a common data model (Section 3.1), (*ii*) the uniform support for queries (Section 3.2), and (*iii*) the managing of constraints (Section 3.3).

3.1. Handling heterogeneous sources

The first part of our approach adresses to the need for integrating several data sources stored in a variety of formats, providing a mechanism to organize and classify such resources. For this purpose, we rely on a common data model for storing the information of an arbitrary number of meta-levels. Heterogeneous sources, like XML or Ontologies can then be integrated by establishing transformations into our common data model (cf. Figure 2). As we will see in Section 4, we have designed an extensible, component-based architecture which permits adding support for new sources – with so called format assistants – externally.

In detail, each file or information source is represented by a Resource, which can be aggregated into Repository objects. Each resource contains a collection of SemanticNode, i.e., entities that are added to account for different technical spaces [25]. In other words, semantic nodes are elements that gather knowledge from (original) source elements and hence, serve as an abstraction for managing heterogeneous information. Resources can be nested to account for hierarchical organization. For example, in MetaDepth and EMF, models and packages are nested, respectively.

Resources, nodes and properties are NamedElements that are identified by their name and can both act as descriptor, i.e., be a type or class, and be described by other elements, i.e., be instances of other elements. Further, our common data model can accommodate instance-relations found in heterogeneous technical spaces, which include (*i*) descriptor-only elements, such as, meta-classes in

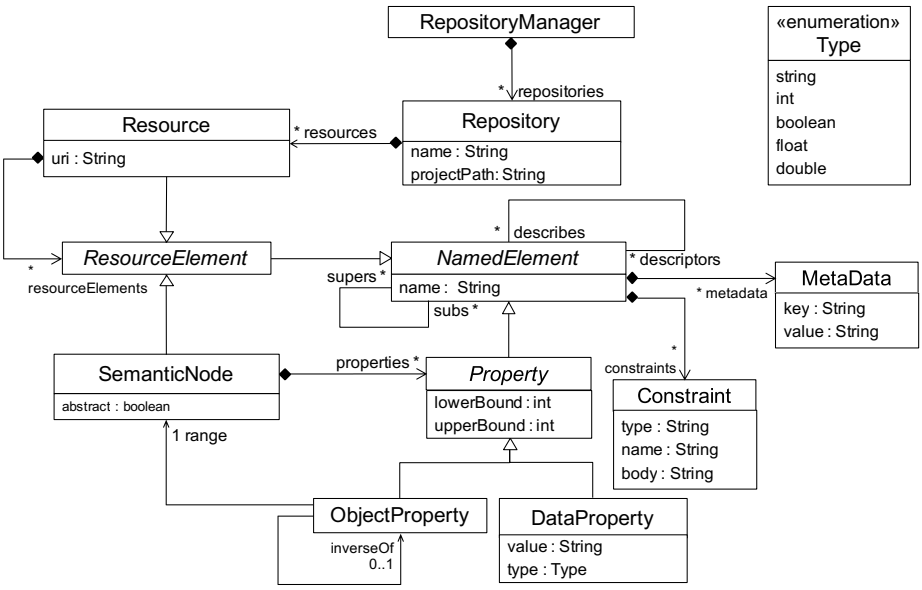

Figure 2: The common data model (package dataModel)

meta-models, (*ii*) described-only elements, such as objects in models, and (*iii*) elements that are descriptors of other elements and are described by others simultaneously, such as clabjects[2] as applied in multi-level modelling [26]. Hence, our data model is meta-level agnostic, as we represent with the same concepts both models and meta-models, classes and objects, and attributes and slots, leading to simplicity and generality [27]. Moreover, our data model can accommodate elements that are described by several elements. Thus, we can accommodate non-exclusive class membership of objects, such as found in Ontologies, and modelling approaches supporting multiple typing, like MetaDepth [28] or the SMOF OMG standard [29].

NamedElements can be associated with MetaData to account for technology-specific details that do not fit into our common data model. For example, when reading an EMF meta-model or a MetaDepth model, it may be necessary to store whether an object property represents a composition or the potency[3] of an element, respectively. Additionally, a Resource, which is also a NamedElement, can manifest conformance relations between artifacts, such as models and meta-models, or XML documents and XML schema descriptions (XSDs), and thus permits representing simple mega-models [30].

SemanticNodes can take part in generalization hierarchies (multiple inheritance

[2]A clabject is a model element that has both type and instance facets and hence holds both attributes, i.e., field types or classes, and slots, i.e., field values or instances.

[3]The potency of an element is represented by zero or a positive number that accounts for the instantiation-depth of an element at subsequent meta-levels [26].

is supported), and be tagged as *abstract*. Generalization hierarchies can be supported at any meta-level (i.e., not only at the class level), to account for approaches where inheritance can occur at the object level [31]. A node is made of a set of properties (DataProperty) and a set of links to other nodes (ObjectProperty), both defining cardinality intervals. Similar to nodes, properties unify the concept of *attribute*, i.e. a specification of required properties in instances, and *slot*, i.e. a holder for values. The common data model supports a range of neutral basic data types (Type enumeration), such as string, int, boolean and double. For generality, the value of the property is stored as a String. Finally, any element can have Constraints attached. The handling of heterogeneous constraints will be explained in Section 3.3.

Table 1 shows how several technologies can be mapped to our data model. We consider the modelling space (in particular the Eclipse Modelling Framework (EMF) [32], a widely used implementation of the Meta Object Facility (MOF) OMG standard [33]), ontologies and XSDs. In all three cases, we show how to map elements at different meta-levels into our common data model. Section 5.3 will assess the generality of our data model by means of an analytical evaluation.

Common Data Model	**EMF**	**OWL**	**XSD**
Meta-level (types)			
Resource	Ecore file/EPackage	OWL file	XSD file
SemanticNode	EClass	OWL Class	xs:element
Property (abstract)	EStructuralFeature	rdfs:domain	xs:complexType xs:element
ObjectProperty	EReference	owl:ObjectProperty	Nested xs:element IDREF attribute
DataProperty	EAttribute	owl:DatatypeProperty	xs:attribute
Property.supers	EClass.eSuperTypes	Inverse of rdfs:subClassOf	xs:element type attribute
Constraint	OCL EAnnotation	N/A	xs:restriction
Model/Data level (instances)			
Resource	XMI file	OWL file	XML file
SemanticNode	EObject	Individual	XML element
ObjectProperty	Java reference	owl:ObjectProperty	Nested xs:element IDREF attribute
DataProperty	Java attribute	owl:DatatypeProperty	XML attribute

Table 1: Mapping different representation technologies to the common data model

EMF supports two meta-levels, and the mapping is direct. Figure 3 shows a schema of how the translation from EMF into our data model is performed. The figure shows on the top that both meta-models (called *ecore* models) and models (typically serialized in XMI format) are transformed into Resources. In this case, the Resource object from the model is described by the Resource of the meta-model. The elements within both meta-models and models follow this translation scheme as well. Both EClasses and EObjects are transformed into SemanticNodes with a suitable descriptor relation, and similar for references and

attributes. At the model level (in the compiled mode of EMF) links and slots are represented as Java fields, whose value is obtained via getter methods.

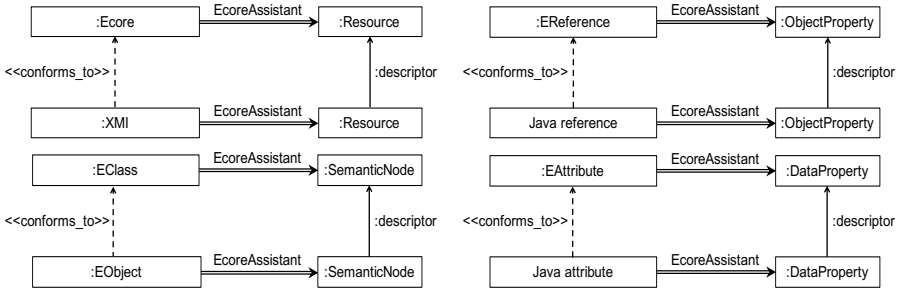

Figure 3: Injecting EMF (meta-)models into the common data model

Figure 4 depicts the translation of XSDs, i.e., acting as language definitions and XML documents. This case is conceptually similar to EMF. The figure shows on the top that an schema, typically serialized in XSD format, is transformed into a Resource of our common data model. Then, as a result of XSDs being described in the XML format, the Resource object from the document is described by the Resource of the schema.

The elements within the schema and the document follow this translation as well. For example, an XML element is transformed into a SemanticNode with a descriptor relation to an xs:element. Moreover, an XML element or XML attribute is transformed either into an ObjectProperty or a DataProperty depending on the type it specifies. In particular, in case of xs:IDREF an ObjectProperty is created, while a DataProperty is created for any other type.

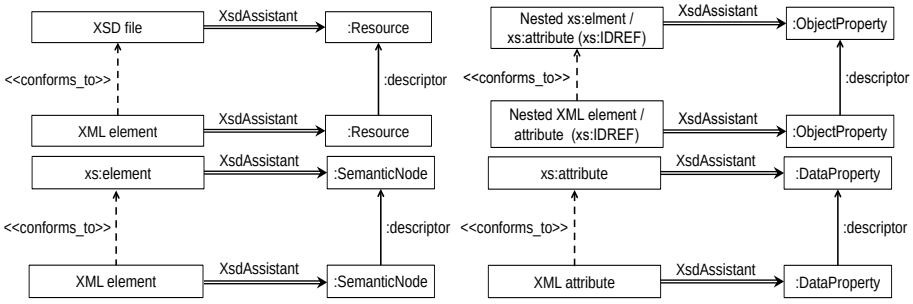

Figure 4: Injecting XML schema descriptions into the common data model

Finally, in the case of Ontologies there are no explicit meta-levels (Figure 5). Then, classes may have several descriptors, and individuals (instances of a class) can have several classifiers, covering the different levels in the representation of concepts. Thus, SemanticNodes and Properties can take part in generalization hierarchies, all of them represented by the taxonomy.

The technical realization of new format assistants will be explained in Sec-

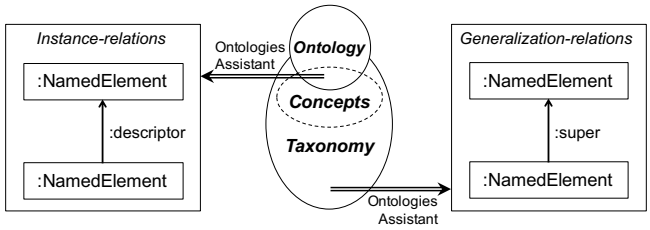

Figure 5: Injecting ontologies into the common data model

tion 4, while Section 5.3 will evaluate the generality of the data model.

3.2. Querying

Once the heterogeneous information is transformed into a common representation, it can be queried in a uniform way. As information conforming to many different schemas may be available in the data model, the query mechanism provided needs to support the flexible exploration of the information gathered. Moreover, oftentimes, the developer may only have a vague idea on what to search, hence a mechanism for inexact matching [34] is needed. This mechanism should be able to provide the user with not only exact matches, but with other related elements as well.

Thus, we refrained from using directly standard model query languages, like the Object Constraint Language (OCL) [35], because they would assume a precise, unique domain meta-model (while may need to query several models conformant to different domain meta-models), and rely on exact matches of queries. Moreover, queries would need to be re-expressed using the meta-model in Figure 2, instead of in terms of the domain meta-models, which may lead to cumbersome expressions. Finally, we need our query mechanism to work at any meta-level.

Figure 6 shows the meta-model describing our extensible query approach. The meta-model provides a mechanism for configuring queries (by specifying the necessary query inputs, class SearchParam), while the actual query results are reified using objects of type SearchResult. The results can be aggregated in groups (through class GroupedSearchResult), or returned atomically (class AtomicSearchResult).

Our approach supports two types of queries: atomic (SimpleSearchConfiguration) and composite (CompositeSearchConfiguration). Atomic queries are configured by declaring a set of SearchParams (representing the expected input parameters from the user), specifying the type of element it searches (filterBy attribute). Composite queries are formed by combining other (atomic or composite) queries, through the and, or and not logical connectives. Composite queries are issued by combining the result sets of previous (simple or composite) queries (reference CompositeSearchConfiguration.inputSearchResults), and may be nested (composition reference CompositeSearchConfiguration.children).

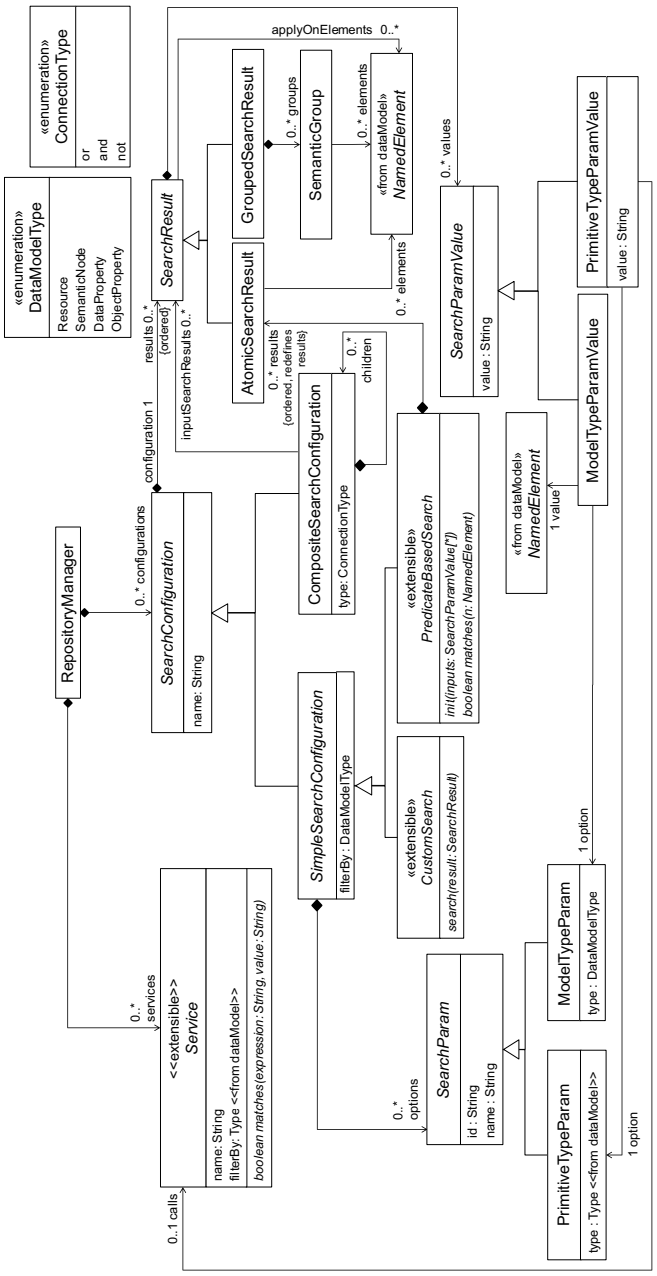

Figure 6: Meta-model of our extensible query mechanism (package queries).

Atomic queries can follow two styles: predicate-based (class PredicateBasedSearch) or custom (class CustomSearch). In both cases, user-defined queries are expected to extend these two abstract classes. In practice, as we will show in Section 4, this is realized by profitting from Eclipse extension points. Their

difference relies on how the iteration is performed: internally (driven by the engine), or externally (driven by the user code); and on the possibility of grouping the results. In practice, predicate-based queries are easier to specify, while custom queries permit more control on how outputs are presented.

Predicate-based queries select their result by providing a boolean predicate, evaluated on NamedElements. Their result is atomic (i.e., not divided in groups), made of all NamedElement objects satisfying the predicate. This kind of queries feature internal iteration. This way, our engine is responsible to traverse the repository and iterativelly pass each object (of the type expected by filterBy) to the matches method. Alternativelly, CustomSearch queries are responsible to both traverse the elements in the repository (external iteration) and select those elements fulfilling the query. This is a more flexible way of querying, which may construct results aggregated in groups.

When a custom query is to be executed, the search method receives a SearchResult object, which initially contains a set of SearchParamValue objects with the input parameter values (as entered by the user), and optionally a set of elements over which the query is to be executed (applyOnElements collection). If applyOnElements is empty, then the query is assumed to search through the whole repository. The input parameters of a search can either be of primitive data type (class PrimitiveTypeParamValue), or model elements of a given kind like Resource, SemanticNode, DataProperty or ObjectProperty (class ModelTypeParamValue). After the execution of the search method the SearchResult must point to the selected element. In the grouped elements output case (class GroupedSearchResult), the search method implementation must decide how to split the members from the elements collection by the definition of SemanticGroups.

Please note that different invocations to the search method results in different SearchResult objects, placed in the results ordered collection. This enables having a history of searches performed, which can be useful in explaning the provenance of the different elements in a model being built.

In a predicate-based query, we rely on a matches method that must evaluate if the list of elements from the input collection belongs to the output (elements collection) or not. In this case, results are saved as AtomicSearchResults. Before the iteration starts, the init method is invoked, receiving a collection of input values (objects of type SearchParamValue), corresponding to the query parameter values input by the user.

3.2.1. Query services

As seen in the meta-model of Figure 6, queries can make use of Services through an extensible mechanism (extending meta-class Service). Services are functions for certain data types from our data model, such as strings or integers. Therefore, services can be used by a particular query on SearchParamValues of a given type. Next we describe some of the most prominent services we provide.

Inexact Matching Searchs that rely on exact string matching are likely to provide poor results, as entities can be named using compound names

("ProcessModel"), or using different derivations (e.g., "processing", "processed"). This way, we provide a means for *inexact matching* [34] to increase the possibilities of finding useful information. The service is based on two techiques: (i) detection of compound names, and (ii) comparing words using their roots. This service is available on search inputs of type String (i.e., when the corresponding PrimitiveTypeParam is typed as String).

Regarding the first technique, it is quite common to find entities in meta-models or ontologies whose name is the concatenation of two or more words, most often in camel case. Our service considers the complete word, and also its different parts in the comparison. Regarding the second technique, comparing word pairs (e.g., "process", "procedure") might throw unsatisfying, or too general, results, even if they belong to the same word sense. For this reasone, the service uses the Porter stemmer algorithm [36], a method to reduce the comparison of two words to their lexical roots.

Word synonym expansion and ranking The exploration of a domain is a difficult task for two reasons: (*i*) in a new domain, with no experience background, a developer may lack the knowledge base about the vocabulary that defines the domain and (*ii*) in a known domain, a developer can lose sight of the main parts of the domain and, as a consequence, build a poor meta-model. Thus, it might be useful to increase the name-based searchs to consider synonyms relevant for the domain. However, words have synonyms with respect to different senses and therefore better search results are obtained by ruling out unrelated senses. For example, the word "glass" has different senses, e.g., to refer to "tumbler or drinking container" and to "lens to aid in vision".

Thus, we offer a service that, given a set of terms, expand them creating a list of synonyms, ranking them according to the relevance for the domain of the input terms. The service is inspired by the Lesk algorithm [37] and evaluates each term in a sentence assuming that every term tends to share a common semantic field with its siblings. For that purpose, we use Wordnet (a lexical database for the English language) [38] to select, among the available entities, the most suitable candidates to match the common semantic fields from an input term list. The list of candidates is obtained by the use of a rule-based system. In the system, we assign points to the whole list of senses provided by Wordnet while they are evaluated to discover whether they fit to the target domain or not.

The points are given by an strategy depending on which rule is more important during the evaluation. With the service we provide a set of strategies but new strategies can be added as well. Next, we present the rules for assigning those points and the strategies defined:

We consider the following entities to calculate the ranking:

- U is the universal set of all possible terms, or words.

- $T = \{t_1, ..., t_n\} \subseteq U$ is a set of input terms, or words, e.g. "process", "activity", "task".
- $S = \{s_1, ..., s_m\}$ is a set of senses, for example the term "process" can refer to "set of actions" (e.g., s_1) or "a series of changes in the body" (e.g., s_2).
- We consider phrases (denoted by p_i), made of sequences of terms. We write $t \in p_i$ to denote that term t occurs in the phrase p_i.

We assume the following linguistic functions and predicates, which can be provided by systems like Wordnet:

- Given a term $t \in U$, function $sense(t)$ returns the different senses of t. For example $sense(process) = \{s_1, s_2\}$.
- Given a sense $s \in S$, function $syn(s)$ returns a set of terms, which are synonyms with respect to the sense s.
- Given a term $t \in U$, function $defs(t)$ returns a set of definitions (a set of sentences) of term t.
- Given a term $t \in U$, function $exs(t) = \{p_1, ..., p_m \mid t \in p_i\}$ returns a set of example sentences, where each p_i contains the term t.
- Given two terms, predicate $syn(t, t')$ holds if both terms are synonyms, independently of the sense.
- Given two terms $deriv(t, t')$ holds if the terms have a linguistic derivative path.

Given a set of terms T, our system returns a set of terms $O = T \bigcup_{t \in T} \{u \in U \mid syn(t, u)\}$, made of the input terms plus their synonyms where the words in O are ranked according to a number of points, given by the following rules:

$\mathbf{R}_1$: Synonyms of input terms

$$\forall t, t' \in T \bullet syn(t, t') \implies points(t') := points(t') + p_1$$

$\mathbf{R}_2$: Synonyms of sense synonyms

$$\forall t \in T,\ \forall s \in sense(t),\ \forall g \in syn(s) \bullet \\ syn(t, g) \implies points(g) := points(g) + p_2$$

$\mathbf{R}_3$: Linguistic derivatives of sense synonyms

$$\forall t \in T,\ \forall s \in sense(t),\ \forall g \in syn(s) \bullet \\ deriv(t, g) \implies points(g) := points(g) + p_3$$

$\mathbf{R}_4$: Synonyms in definitions

$$\forall t \in T,\ \forall s \in sense(t),\ \forall g \in syn(s),\ \forall p \in defs(t) \bullet \\ g \in p \implies points(g) := points(g) + p_4$$

$\mathbf{R}_5$**:** Synonyms in examples

$$\forall t \in T,\ \forall s \in sense(t),\ \forall g \in syn(s),\ \forall p \in exs(t) \bullet$$
$$g \in p \implies points(g) := points(g) + p_5$$

Several strategies can be used to assign points to every rule:

- All the same: Each $\mathbf{R}_i$ receives the same quantity of points (p_i).
- Synonyms first: $\mathbf{R}_1$ and $\mathbf{R}_2$ receive more points than the rest of rules.
- Definitions first: $\mathbf{R}_4$ receives more points than the rest of rules.
- Examples first: $\mathbf{R}_5$ receives more points than the rest of rules.
- Custom: A distribution based on a custom criteria.

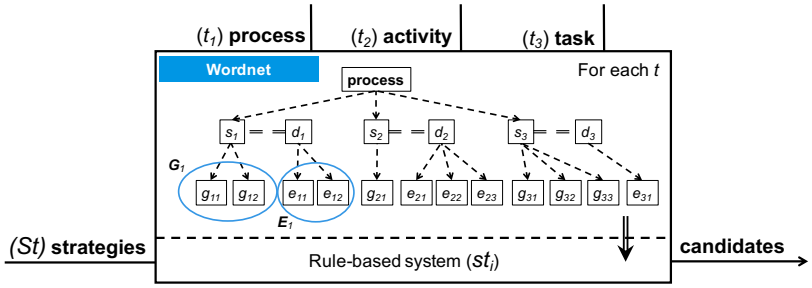

Figure 7: Example execution of the synonym expansion and ranking service

Example. Figure 7 shows an execution schema for the service. Firstly, a set T of terms is received by the service. For each term (t_i), a tree of senses (s_{ij}), definitions (d_i), examples (e_{ij}) and synonyms (g_{ij}) is formed, using information from Wordnet. The set of rules are applied for each term according to a selected strategy (st_i). For example, for the list of terms $T = process, activity, task$ for a custom strategy that assigns 1000 points to the value p_1, 80 to p_2, 20 to p_3, 100 to p_4 and 20 to p_5, the following output list of ranked candidates is obtained:

1000 task
1000 process
1000 activity
320 job
320 chore
300 summons
260 body process
260 bodily process
260 bodily function
240 procedure
200 treat
200 physical process
200 outgrowth
200 appendage
180 natural process
180 natural action
100 work on
100 work

100 unconscious process
100 swear out
100 sue
100 serve
100 march
100 litigate
100 action
0 ~~undertaking~~
0 ~~project~~
0 ~~operation~~
0 ~~mental process~~
0 ~~labor~~
0 ~~cognitive process~~
0 ~~cognitive operation~~
0 ~~activeness~~

where the service discards the words with no points.

Numeric intervals For numeric properties, instead of writing concrete values, it is possible to enter intervals (e.g., [1..5] or [2..*]) to achieve more flexible queries.

3.2.2. Examples

In this section we present two examples, illustrating the query extensibility mechanisms and the services provided. The list of all available queries is shown in Table 9 in the appendix.

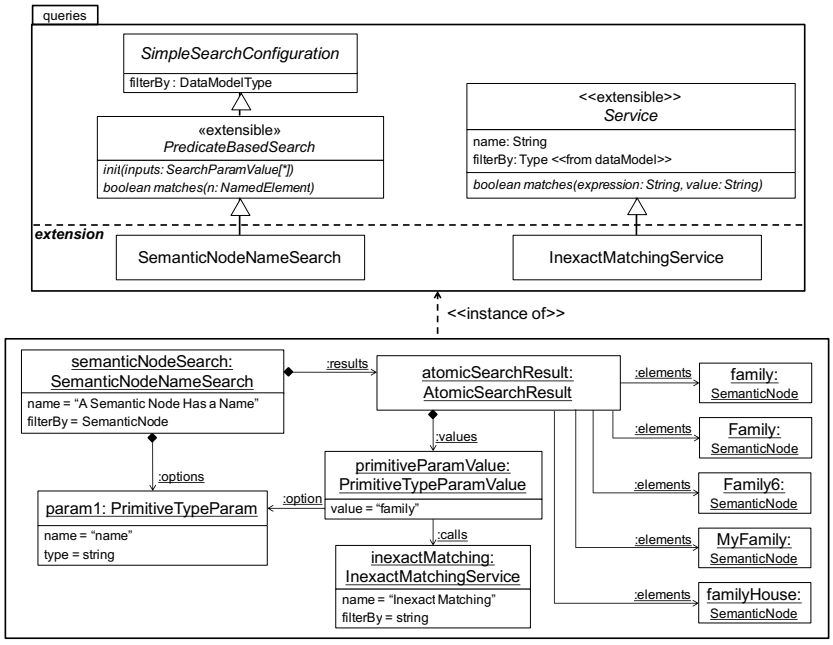

Figure 8: An instance of a predicate-based search (PredicateBasedSearch) with an atomic result (AtomicSearchResult)

Figure 8 depicts a predicate-based search that gathers the results atomically. The upper part of the figure represents an excerpt of the queries package shown in Figure 6, which is extended with a query (SemanticNodeNameSearch) and the "Inexact Matching" service explained in Section 3.2.1.

The lower part of the figure shows an instance model containing the parameter of the search (object PrimitiveTypeParam), its input value (object PrimitiveTypeParamValue) and the search results (semantic nodes referenced from object searchResult). Overall, the input value "family" is received by the search, which is then passed to the service. The service checks whether the attribute value of the semantic node matches with the expression or not. Finally, the set of selected semantic nodes are attached to the AtomicSearchResult object.

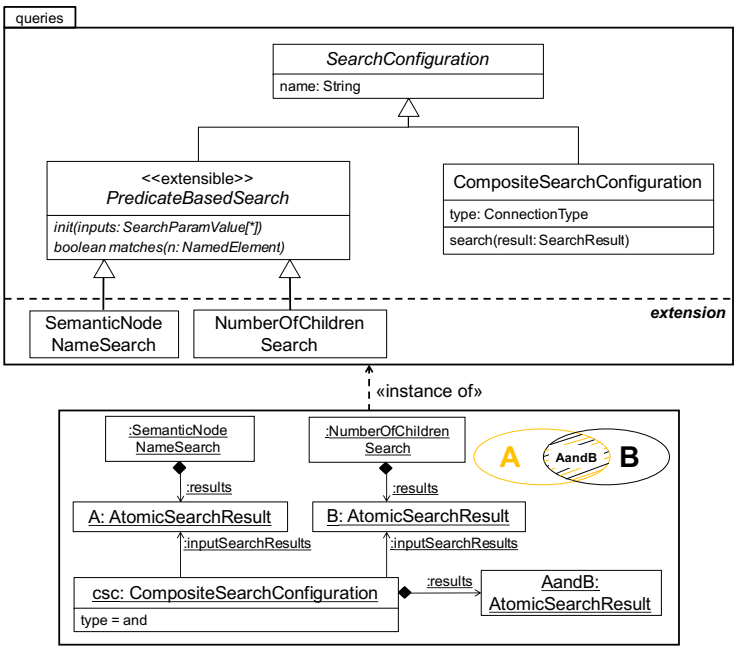

Figure 9: An and-type composition (CompositeSearchConfiguration) of two searches

Next, Figure 9 shows a composite search made of the conjunction of the search result of two queries: the SemanticNodeNameSearch query and NumberOfChildrenSearch (which searches for nodes with more than a number of children through a generalization hierarchy, cf. Table 9). The composite query returns SemanticNodes belonging to both search results, as schematically depicted by the Venn diagram in the rop-right corner of the lower model.

3.3. Handling constraints

In our data model, SemanticNodes are used to store the knowledge found in an element, including properties and links to other nodes. Additionally, they may include restrictions to be satisfied by their instances.

For example, in meta-models, a class may contain OCL invariants that all objects of such class should obey. Other modelling technologies, like MetaDepth allow attaching constraints to whole models [12]. Similarly, elements in XML schemas may contain restrictions on properties of a given data type. These may be used to define patterns on strings (regular expressions), restrictions on the string length (min/max/exact length), and handling of white spaces, among others. While OWL does not directly support integrity constraints, extensions have been proposed for this purpose [39, 40].

Thus, our data model includes a facility to store and interpret heterogeneous constraints. Constraints can be attached to any NamedElement, which includes Resources, SemanticNodes and Properties, covering the main scenarios in the different technical spaces. Constraints have a type identifying the kind of constraint (e.g., OCL), a name, and a body.

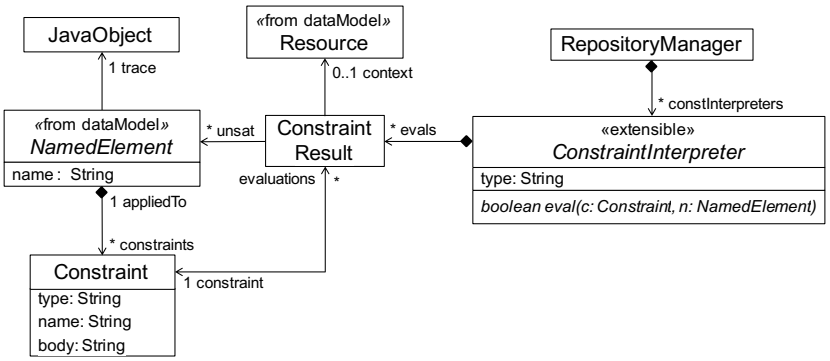

Figure 10: Meta-model of the extensible constraint handling mechanism.

In order to support evaluation of heterogeneous constraints (OCL, XML schema restrictions, etc) our approach is extensible. This way, constraint interpreters can be incorporated by extending class ConstraintInterpreter, declaring the constraint types they can handle, and implementing the evaluate method. The method receives a constraint and an instance of the element the constraint is attached to, and returns a boolean indicating if the element satisfies the constraint. As we will see in Section 4, the addition of constraint interpreters is done through an Eclipse extension point. Similar to query results, constraint results are reified using ConstraintResult objects, which hold elements that do not satisfy the constraint, and in addition organizes results in the context of the enclosing Resource.

4. Architecture and tool support

We have realized the previously presented concepts in a tool called Extremo. It has been implemented as an Eclipse plugin, is open source, and is freely available at `http://miso.es/tools/extremo.html`. The web page in-

cludes videos, screenshots and installation details. A schema of Extremo's architecture is shown in Figure 11.

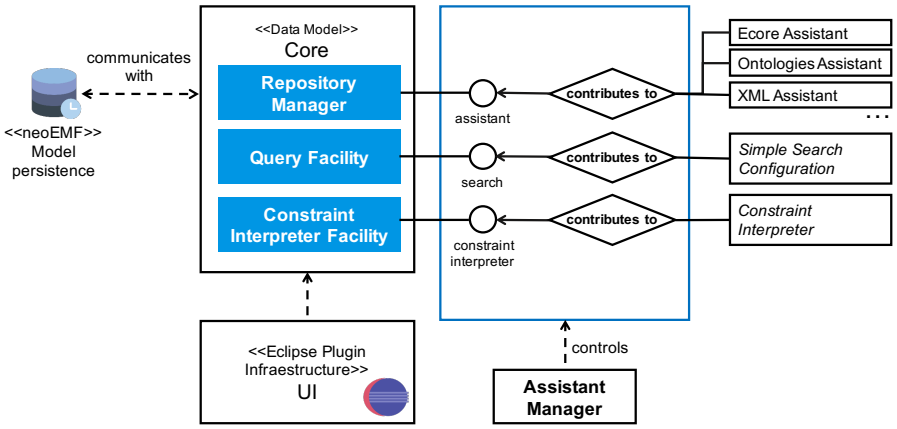

Figure 11: Architecture of Extremo

Extremo is made of a Core component, which provides support for the common data model and includes subcomponents for the query and constraint handling facilities. The Core component can be extended in different ways, e.g., to provide access to heterogeneous information sources, as shown in Figure 11. This extensibility is realized through Eclipse extension points. These are interfaces that can be implemented by external components to provide customized functionality.

To allow for scalability, the repository is persisted using NeoEMF [41], a model persistence solution designed to store models in NoSQL datastores. NeoEMF is based on a lazy-loading mechanism that transparently brings into memory model elements only when they are accessed, and removes them when they are no longer needed.

The UI component permits visualization and interaction with the resources and query results. A set of views, commands, wizards and actions has been defined to control the access to the repository, the list of query results and the constraint validation facilities. By extending this component, it is possible to integrate Extremo with external modelling tools.

Next, we will detail the structure of the Core subsystem in Section 4.1, while the UI subsystem will be described in Section 4.2.

4.1. The Core subsystem

Figure 12 shows the main components of the Core subsystem: (*i*) a Repository Manager, which controls the access to the common data model and assists in the process of incorporating a new technological space; (*ii*) a Query Facility, which supports our query mechanisms in an extensible way; and (*iii*) a Constraint Interpreter Facility, which provides flexible support for different constraint formats and interpreters.

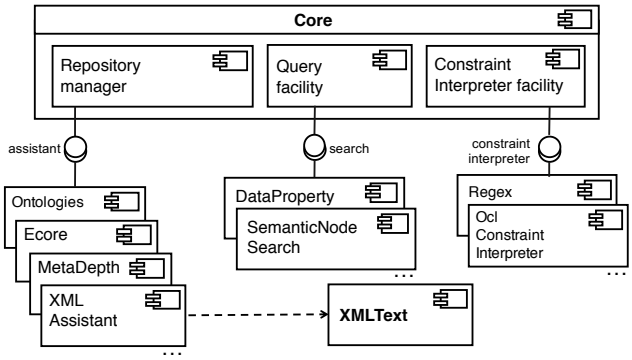

Figure 12: Architecture of the Core component

For each part, a set of extension points have been defined, and the figure shows some example implementations of these. The first extension point (assistant) permits adding support for new data formats. It requires implementing a mapping from the format-specific structure, such as XML, to the common data model, as described in Section 3.1.

We predefined a set of assistants as well as a framework for their creation, which permits their conceptual organization as model-to-model transformations. Hence, we provide an abstract class with a number of methods, which need to be overriden for the particular assistant, and act as rules in a transformation. In these methods, it is possible to create one or more elements in the common data model, hence supporting one-to-many and many-to-one mappings.

To facilitate the construction of new assistants, it is possible to define class hierarchies, and hence reuse import functionality. In addition, our scheme can profit from existing transformations between two specific technical spaces. For example, if one had a transformation from XSD to Ecore, and an assistant for Ecore (translating Ecore to Extremo's common data model), then an assistant for XSD can be built by first translating into Ecore and then invoking the Ecore assistant. This is the way we implemented the XsdAssistant (see Figure 4) [42, 43, 44].

The second and third extension points provide extensibility for queries. Conceptually, user defined queries extend the meta-model of Figure 6. The extensions allow defining custom, predicate-based, and composite queries by subclassing from the corresponding classes in Figure 6. Finally, the last extension point permits contributing support for evaluating new types of constraints, extending the meta-model in Figure 10.

4.2. The UI subsystem

Figure 13 shows the architecture of the UI subsystem. It is made of contributions to the Eclipse infrastructure to visualize and interact with the repository, to visualize the search results and the list of elements that satisfy a constraint.

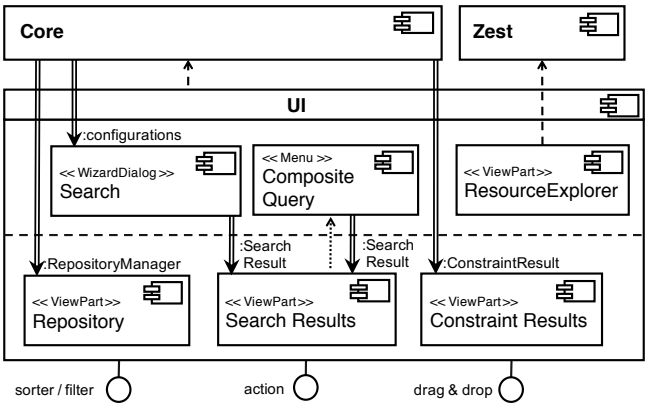

Figure 13: Architecture of the UI component

The UI subsystem is composed of: (*i*) a Search wizard dialog, that receives the list of search configurations; (*ii*) a Resource Explorer, a mechanism based on Zest (see `https://www.eclipse.org/gef/zest/`), a component for graph-based visualization, which provides support for filtering and layouts; and (*iii*) a set of view parts, that reflect the current state of the repository set model, the query results (as instances of the class SearchResult) and the constraints validation. All of them can be sorted or filtered by means of an extension point.

As an example, Figure 14 shows Extremo in action. In particular, it shows the query dialog by which any of the defined queries can be selected, input values can be given to each of its input parameters, and services can be selected depending on the types of the parameters. In the lower part, the figure shows the repository view, with some resources and their content; and the search result view, which permits browsing through the query results. It must be noted that semantic nodes in the repository view have distinct icons, depending on whether they contain data, object properties or constraints, and on whether they are types, instances or both.

Figure 15 shows the resource explorer. In particular, it shows on the right an instance of our common data model and the relationships between nodes. Since the resource explorer is based on a component for graph-based visualization, the SemanticNode instances are represented as nodes and the ObjectProperty instances are represented as edges of the graph. The left part shows how the resource explorer can be invoked from every resource with a action contribution to the pop-up contextual menu of the repository view.

The UI subsystem has been designed so that it can be flexibly integrated with external modelling tools. For that purpose, two extension points have been defined. The first one enables the addition of an action to the view parts, as a reference in the contextual menu and the toolbar. The other extension point enables the drag operation from the views and the dropping of the selected information into any graphical editor based on Graphical Editing Framework [45].

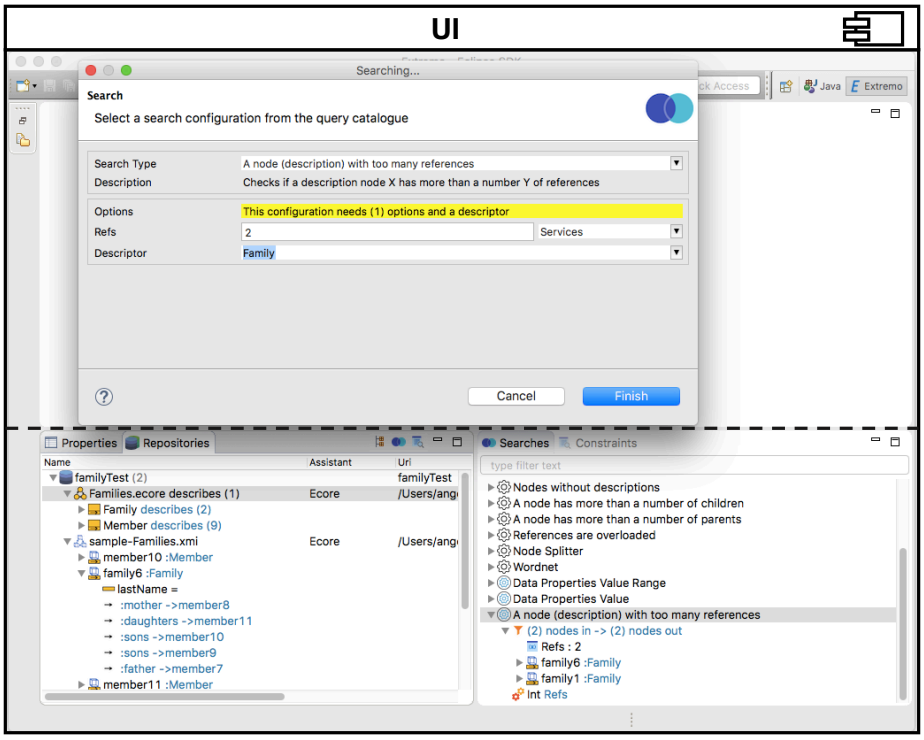

Figure 14: EXTREMO in Action: Search Wizard Dialog and views

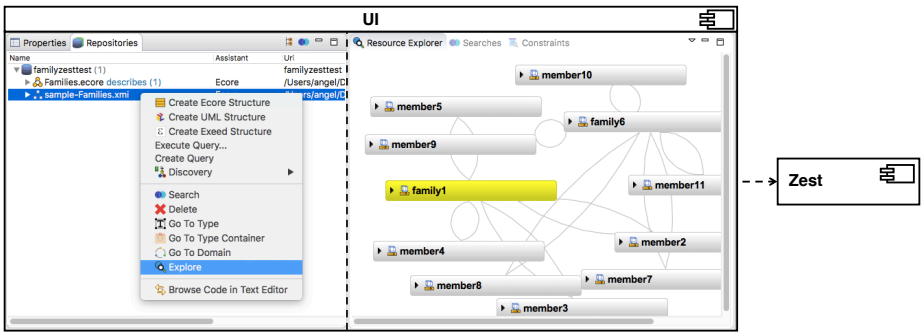

Figure 15: EXTREMO in Action: Resource Explorer

This is the underlying framework of Eclipse graphical editors. We will explain these two integration styles in the next two sections.

4.2.1. Integration with a modelling tool by means of actions

In this section we present the integration of the core components of EXTREMO with a modelling tool using an action contribution to the view parts of

the UI subsystem. We will illustrate this section with the UML model editor[4], a solution based on a TreeEditor. This kind of editor is an extensible part of the Eclipse infraestructure.

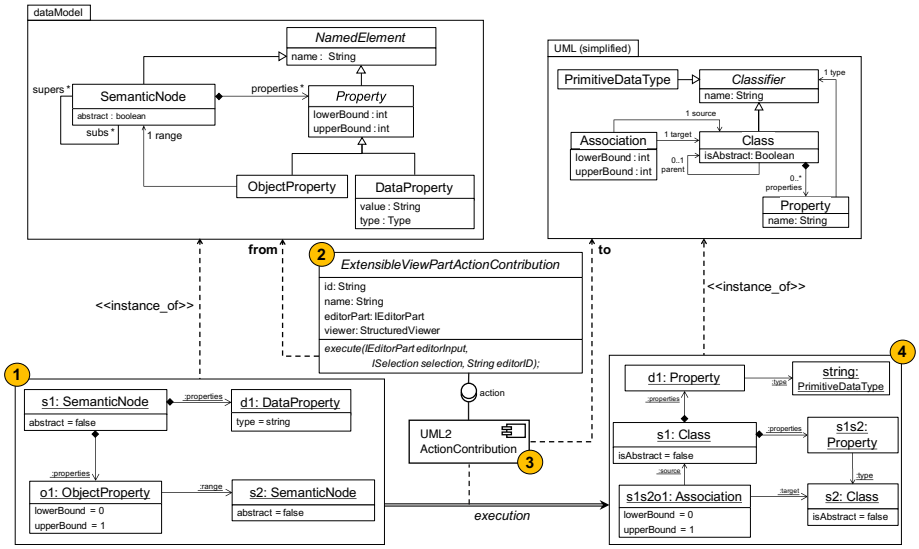

Figure 16: Example: action-based integration

Figure 16 illustrates the working scheme of this integration style, where an instance of our data model (Figure 2) selected by the user (e.g., via a query) is incorporated into a UML model [4]. The elements selected by the user (label 1) are two non-abstract SemanticNode objects. The first one contains a DataProperty (d1, typed as string) and an ObjectProperty that refers to the second SemanticNode. The addition of a new action contribution is enabled by an extension point. The action contribution provides the mapping between our common data model and the intrinsic model of the TreeEditor. The action contribution must extend the ExtensibleViewPartActionContribution class (label 2) and implement an abstract method that creates the instances of the target model, conceptually working as a model-to-model transformation (label 3). Finally, a UML model is created (label 4). The initial two SemanticNode objects are mapped into two non-abstract Class objects, the initial DataProperty object is mapped into a Property object and the initial ObjectProperty object is mapped into an Association object.

Figure 17 shows how this is achieved in practice. The repository view of Extremo is shown in label 1. In this view, the user can select a number of elements that are to be incorporated into the UML model (editor shown in label 3). For this purpose, a contextual menu offers the different action contributions (label 2). The UML editor in the figure shows already the Ecore elements

[4]UML2-MDT, `www.eclipse.org/modeling/mdt`

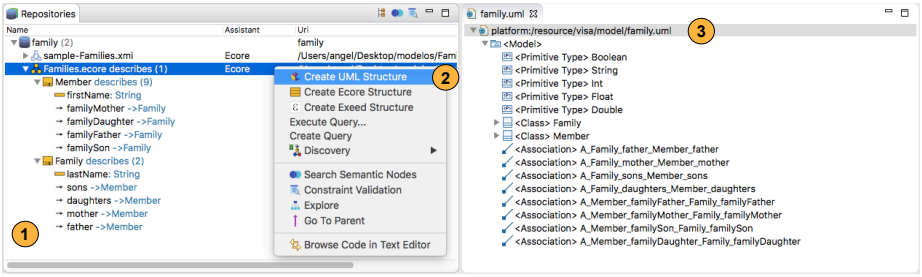

Figure 17: Integrating Extremo with the UML2 modelling tool using an action contribution

incorporated into the UML model.

4.2.2. Integration with a modelling tool by means of drag and drop

The second integration style enables the addition of a drag operation from the views and the dropping of a set of NamedElements into a graphical editor based on Graphical Editing Framework [45], the underlying framework of Eclipse graphical editors.

We will illustrate this section with DSL-tao [19], an Eclipse plugin for the construction of DSLs using a pattern-based approach. The underlying representation of DSL-tao is Ecore. Thus, in this case, Figure 18 shows an example that transfers an instance of our data model (Figure 2) into an instance of Ecore.

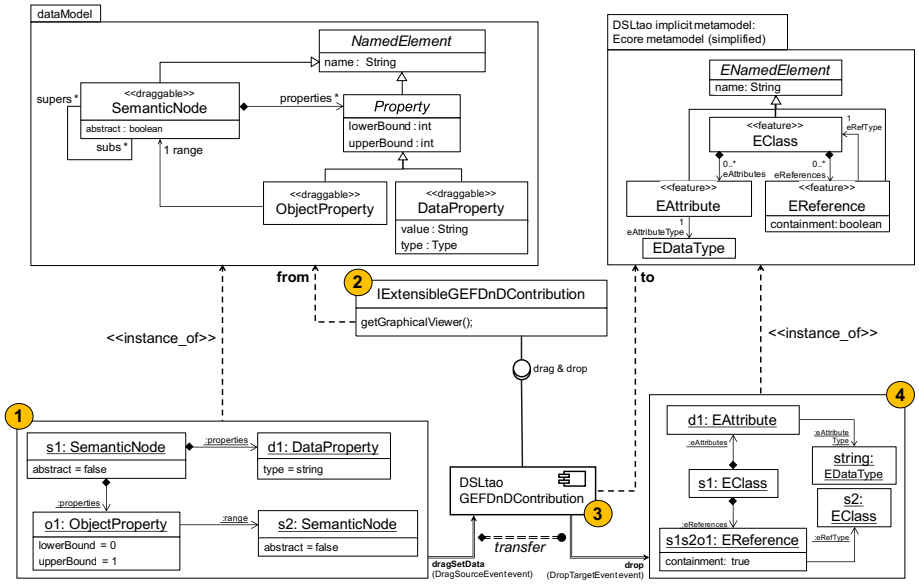

Figure 18: Example: drag and drop based integration

As in the previous section, the portion of the model selected by the user

(label 1) contains two non-abstract SemanticNode objects. The first one contains a DataProperty (d1, typed as string) and an ObjectProperty that refers to the second SemanticNode. The addition of a new dropping contribution is enabled by an extension point. The dropping contribution provides the transfer of a set of NamedElements selected from the views and the catching of the drag and drop event. The dropping contribution must extend the IExtensibleGEFDnDContribution interface (label 2) and implement a method resolving the graphical viewer editor that receives the selected elements (label 3). Finally, an Ecore model is created (label 4). The initial two SemanticNode objects are mapped into two EClass objects, the initial DataProperty object is mapped into an EAttribute object and the initial ObjectProperty object is mapped into an EReference object.

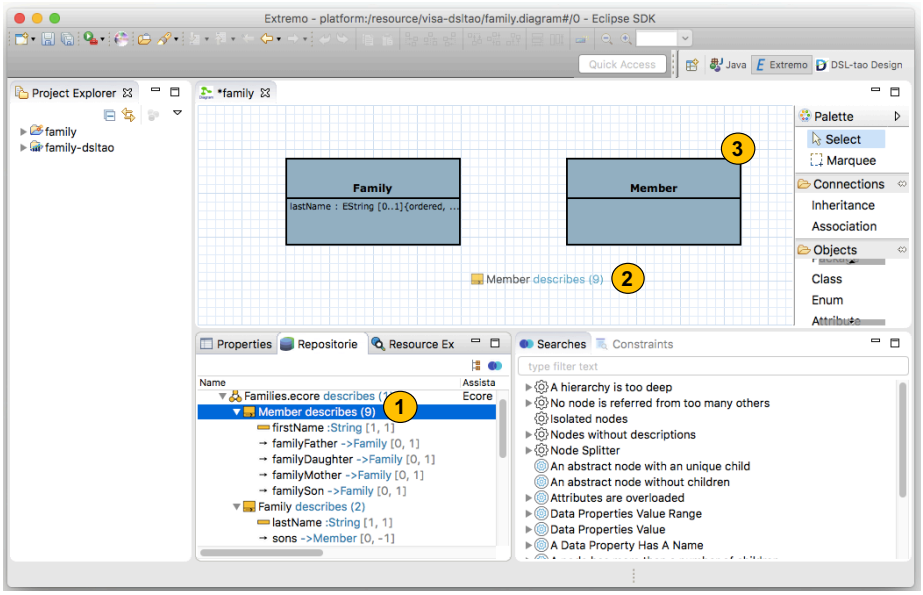

Figure 19: Integrating Extremo with DSL-tao using a drag and drop contribution

Figure 19 shows how this is achieved in practice. The Figure shows the main canvas of DSL-tao, which permits building graphically a meta-model using the palette to the right. The bottom of the figure shows the views contributed by Extremo. In this case, the user may initiate a drag from some model element in the view and drop it into the canvas (labels 2 and 3). In the Figure, the user has incorporated class Member into the meta-model, while it is also possible to incorporate attributes and references.

5. Evaluation

We present an evaluation of our approach under different perspectives. We start by describing the research questions we address in the evaluation in Sec-

tion 5.1, followed by three different evaluations in Sections 5.2, 5.3 and 5.4, and finishing with a discussion of threats to validity in Section 5.5.

5.1. Research Questions

By conducting this evaluation, we aim at solving the following research questions:

RQ1: How useful is Extremo *to solve practical problems in Language Engineering?.*
In order to assess the usefulness of Extremo, we will use two demonstration cases [46]. In both of them, Extremo was used to support the development of complete DSL implementations for process modelling in the immigration domain (Section 5.2.1) and modelling of production systems (Section 5.2.2).

RQ2: How capable is Extremo *to represent information coming from different technological spaces?.*
As previously described, the range of formats currently supported by Extremo can be extended. This way, Section 5.3 presents an analytical evaluation to assess how well Extremo's data model covers the space of possible information description approaches.

RQ3: How integrable is Extremo*?.*
One of the salient features in Extremo is the possibility of integrating it with third-party tools. Having already successfully integrated a number of tools with Extremo, we intend to assess the reach and limitations of the integration mechanism. This question is approached in Section 5.4.

5.2. Evaluating usefulness

In this section, we evaluate usefulness by presenting two demonstration cases. The first one is on process modelling in the immigration domain (Section 5.2.1), while the second one is on modelling of production systems (Section 5.2.2). In the cases, Extremo was used in combination with two different modelling tools (DSL-tao and the Ecore tree editor), while assistants for four different technologies were used (Ecore, Ontology, CSV and XML). The purpose of the cases is evaluating the power of combining heterogeneous information sources for language engineering.

Characteristics of the two selected cases are (*i*) they cover both structural and behavioral modelling languages, (*ii*) they combine information from standardized modelling languages and from very focused domain-specific languages, and (*iii*) modelling languages are integrated with small to large domain descriptions coming from different technological spaces. They differ in the variety of technical spaces and the size of resources being considered.

For both cases we will highlight the use of our framework over a sequence of phases, which include (*i*) the scope of the language and an example of model, (*ii*) the collection of resources required, (*iii*) the resource import and querying of the common data model, (*iv*) the construction of a language meta-model and creation of instances, and (*v*) the results obtained.

5.2.1. Immigration process modelling

In the first demonstration case, we present the construction of a DSL for describing administrative e-Government processes for immigration.

Scope of the Language. We will construct a language for the modelling of processes to issue visas according to the American regulation[5], called IWML. The language will include elements showing the variety of offices, concepts from the domain of immigration and agents involved in the process and generic elements of workflow languages like BPMN [47].

Example Model. Figure 20 shows an example IWML model, which contains (*i*) domain-specific knowledge related to the agents (DOS, DHS, Medical Service, User), and the artefacts (the I-130 form, the variety of Visas and the Payment method) involved in the process, (*ii*) specific activities needed in immigration processes, such as ≪Application Process≫, ≪Date Selection≫, or ≪Interview≫, (*iii*) more generic elements – such as tasks, events, and gateways – typically found in process modelling languages. In the figure, a User is required to fill in a standard I-130 form and to set a date for an interview with the embassy personnel. Then, the user is interviewed by the US Department of Home Security (DHS) and they perform a study of the application submitted. Afterwards, the Department of State (DOS) validate the data. In a parallel process, the candidate is expected to go through a Medical Examination. Once both the ordinary verifications and the medical report have been checked, the document is dispatched. Depending on the characteristics of the applicant various types of visa should be issued. In this case, a E1 type Visa is approved and consequently issued; otherwise, the document is denied.

Resource Collection. In order to gather all the required elements we defined a set of repositories and resources according to the data model shown in Figure 2. A list of resources were taken from: (*i*) the Object Management Group (OMG)[6] (`http://www.omg.org/spec/`); (*ii*) Ecore repositories, such as the ATL zoo (`http://web.emn.fr/x-info/atlanmod/`), which includes meta-models for BPEL, DoDAF, Gantt, ODP, SPEM and some others; and (*iii*) Ecore files available on the OMG repository with standard meta-models, such as BMM, BPMN, CMMN, DMN, SBVR and SPEM. For the domain-specific concepts, open resources were required. In our case, we took CSVs from the U.S. Open Data Portal (`https://www.data.gov/`) and US e-Government domain ontologies (`http://oegov.org/`), which are available in OWL and RDF formats. This second repository contains ontologies that model the set of United States government facets. These include the Government ontology, the U.S. Government ontology,

[5] `https://www.usa.gov/visas`

[6] The OMG is the standarization body behind many modelling standards such as UML, SysML, MOF or BPMN.

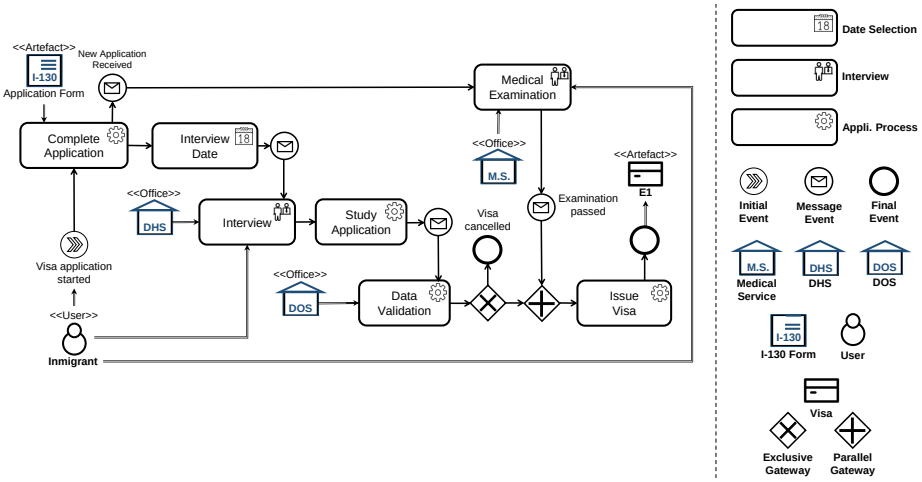

Figure 20: Example process for issuing a visa with IWML (left) and legend (right)

the Department of Homeland Security ontology and Department of State ontology.

Table 2 summarizes the number of instances of the resource collection by focusing on the meta-level and, in the case of the ontologies, taking into account also the individuals. Even though in this case the resources imported were not large (this will be evaluated in the second case study) we can appreciate that EXTREMO was able to gather information coming from different technological spaces. The total size of the reused artefacts amounted to around 2,4 MB of data.

Ontology concept	**Number of instances**
OWL file	22
OWL Class	141
Individuals	1118
owl:ObjectProperty	35
owl:DatatypeProperty	91

EMF concept	**Number of instances**
Ecore file/EPackage	30
EClass/EDataType	2484
EReference	892
EAttribute	326
OCL EAnnotation	0

Table 2: Number of collected instances of different Ontology and EMF-concepts, respectively.

Resource Import. We imported the Ecore meta-models taken from the OMG and the ATL zoo by the application of our EcoreAssistant (cf. Section 3.1) and the domain-specific concepts by applying our OntologyAssistant and the CsvAssistant.

Meta-Model Construction. The meta-model was developed using DSL-tao, integrated with EXTREMO following the approach described in Section 4.2.2

by means of a drag and drop extension point. Figure 21 shows a moment in the construction of the meta-model. In particular, it shows the Eclipse IDE with the EXTREMO perspective open (label 1), which includes the *Resource Collection* previously mentioned in the repository view (label 2) and the BPMN.ecore resource visualized in the resource explorer (label 3).

The meta-model construction phase involved some of the queries listed in Table 9, such as finding types of forms, organizations and gateways. Once a query has been issued, the resulting elements can be highlighted on the original resource (label 4). Finally, a set of semantic nodes can be dropped over the DSL-tao design diagram (label 5).

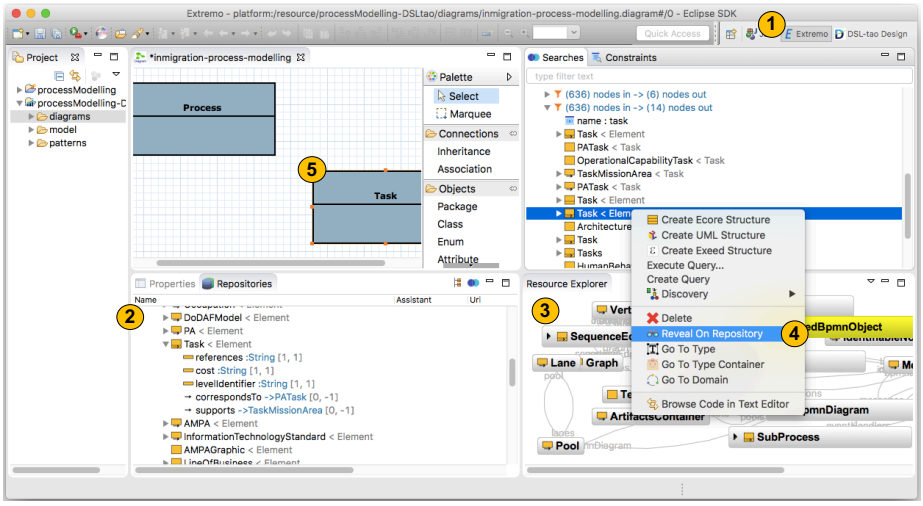

Figure 21: Integration of EXTREMO with DSL-tao (from the perspective of the case study)

Result. Figure 22 shows the final result obtained. In detail, IWML has elements originating from the set of meta-models that describe workflows, such as Gateways, Events or Task. Moreover, IWML is composed of some domain-specific concepts taken from the set of domain ontologies, like DOS, DHS or MedicalService; CSVs, such as the type of Visas or the set of Users. Roughly 22% of the classes in our solution have been obtained from different ontologies and CSVs, 48% of the classes have been obtained by combining different representations of process modelling meta-models and the rest (30%) have been added by hand taking information from the government websites. This suggests that EXTREMO is useful as a help in the construction of a complex meta-model, as we were able to build the meta-model by reusing elements from different heterogeneous information sources.

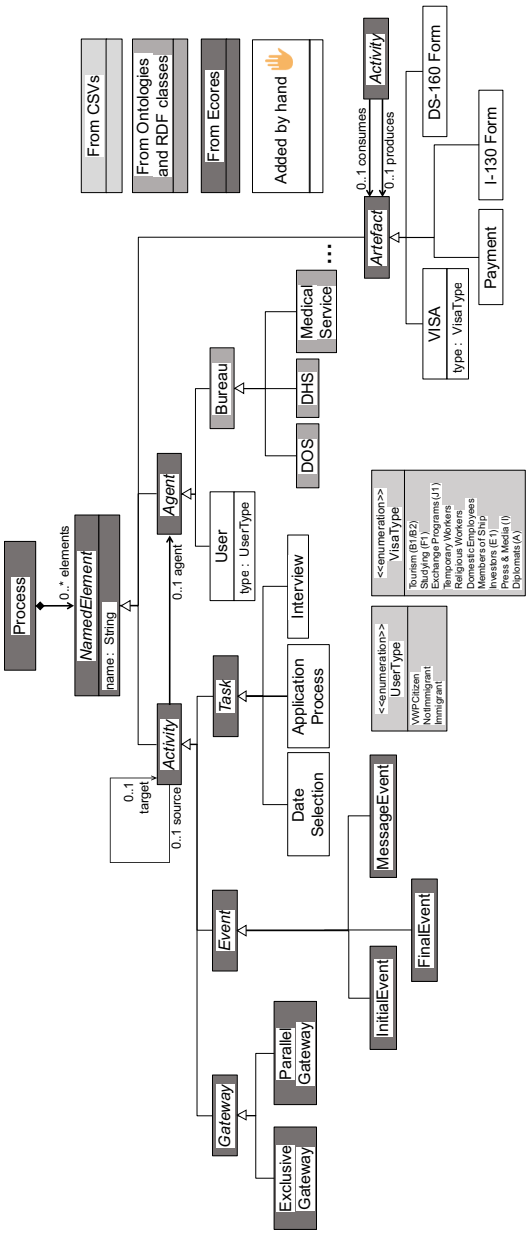

Figure 22: Excerpt of the processes for immigration meta-model.

5.2.2. Industrial production system modelling

In the second case, we present the development of a DSL for industrial production systems to enable the construction of models that conform to an

interoperable cross-industry standard for products and services called eCl@ss[7]. The goal of this case study is to reduce the number of potential candidates for conveyor-belt system-components that are available in the eCl@ss-standard, and thus to conveyor-belt system-modellers, by applying EXTREMO for constructing the desired language.

Scope of the Language. We construct a language for the modelling of production systems conforming to the eCl@ss-standard called EPML. The language includes elements from conveyor-belt systems that must fulfill the constraints imposed by the eCl@ss-standard and the GEMOC initiative[8], such as the Signal Process Modelling Language (SigPML)—a DSL dedicated to data flow processing.

Example Model. Figure 23 shows an example EPML model, which contains (*i*) electrical drives (DC Engine), (*ii*) communication cables or ready-made data cables that represent SigPML connectors, (*iii*) PC-based controls or field buses, i.e., decentralized peripherals, which represent controls, (*iv*) controls in terms of inductive proximity switches, and (*v*) sets of rectangular industrial connectors, which represent connector systems. In the figure, a DC Engine is connected to a Fieldbus through a communication cable and a data cable with a set of industrial connectors. The Fieldbus also has a connection with a Proximity Switch through a communication cable and a data cable using the industrial connectors that the Proximity Switch has.

Resource Collection. In order to gather all the required elements we define a set of repositories and resources according to the data model shown in Figure 2. The resources were taken from: (*i*) the GEMOC initiative, such as SigPML defined in form of an Ecore meta-model; (*ii*) the eCl@ss-standard, defined in form of several XML schema definitions (XSDs) and XML instances.

Table 3 summarizes the number of instances of the eCl@ss-standard as well as the SigPML by focusing on the meta-level. However, domain-specific concepts in the eCl@ss-standard measure a substantial size, i.e., only the basic and advanced specifications in the English language consist of 41,000 product classes and 17,000 properties, which amount to 15.5 Gb of data. In this situation, extracting desired concepts requires the manual examination of a vast amount of resources as well as their re-implementation by a target DSL. Moreover, any update that is performed on eCl@ss-standard resources, may also impact the implementation in the target system and involve complex and time-consuming maintenance tasks. In an effort to counteract such limitations we apply the EXTREMO framework as follows.

[7] An ISO/IEC-compliant international standard for the unified and consistent definition and classification of products, materials, and services alongside typical supply chains by the use of commodity codes.

[8] `http://www.gemoc.org`

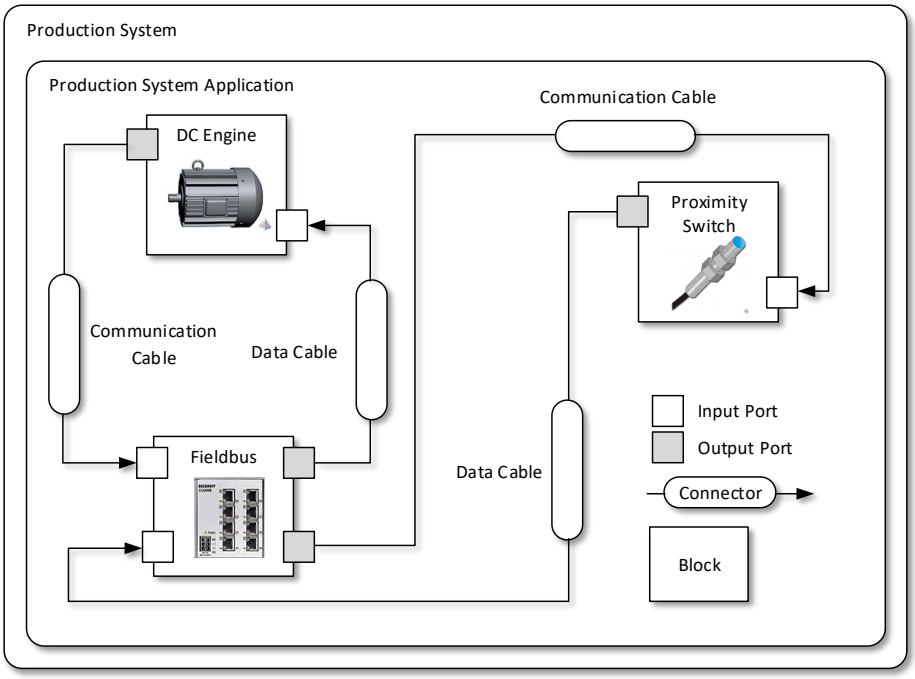

Figure 23: Abstract graphical representation of the conveyor-belt production system model.

XSD concept in eCl@ss-standard	**Number of instances**
XSD file	30
xs:element	960
xs:element IDREF attribute	110
xs:attribute	104
xs:restriction	84

EMF concept in SigPML	**Number of instances**
Ecore file/EPackage	1
EClass	14
EReference	18
EAttribute	10
OCL EAnnotation	0

Table 3: Number of collected instances of different XSD and EMF-concepts, respectively.

Resource Import. First, we import the resource collection by the application of the EcoreAssistant and the XsdAssistant. The XsdAssistant reuses functionality of the XMLIntellEdit framework [44]—composed of XMLText [42] and IntellEdit [43]. The XMLText framework transforms XML-artifacts, i.e., XSDs and XML instances, to corresponding MDE-artifacts, i.e., Ecore meta-models and conforming models. Then, the EcorAssistant is used to map the MDE-artifacts into the common data model.

Meta-Model Construction. Next, we employ the Extremo Eclipse perspective as well as the Sample Reflective Ecore Model Editor, integrated with

EXTREMO following the approach described in Section 4.2.1. Figure 24 shows a moment in the construction of the EPML meta-model. In particular, it shows the set of resources in the repository view (label 1) and the EXTREMO functionalities involved in the meta-model construction phase for querying (label 2), traversing (label 3), and applying desired concepts from the imported repositories (label 4). For example, available concepts that represent an electrical drive in the eCl@ss-standard are gathered by issuing an EXTREMO query for retrieving semantic nodes that are named "engine" and then used for creating corresponding concepts in the EPML meta-model. EXTREMO traversal-functionalities, such as Reveal On Repository, Go To Type, and Go To Domain, are employed for gathering respective classes, which are referenced as super-types and thus enforce conformance to the eCl@ss-standard.

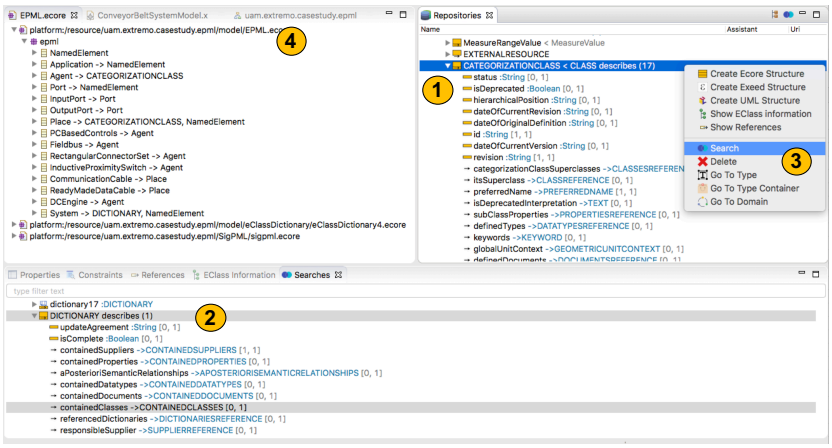

Figure 24: Integration of EXTREMO with the Sample Reflective Ecore Model Editor (from the perspective of the case study)

The final result of the EPML meta-model construction process is depicted in Fig. 25. In detail, the EPML data flow process elements originated from the SigPML (in dark-grey) such as System, Application, Block, Connector, and Port. Moreover, EPML is composed of several eCl@ss-standard concepts (in light-grey), which include (*i*) electrical drives, (*ii*) cables, (*iii*) controls, (*iv*) binary sensors, i.e., safety-related sensors, and (*v*) connector systems. For example, the eCl@ss-standard CATEGORIZATIONCLASS represents the super-type of Block and Connector in EPML. Additionally, subtypes of Block and Connector are also instances of CATEGORIZATIONCLASS in the eCl@ss-standard. As a result of distinguishing specific instances of categorization-classes adds additional EPML-specific semantics that gather concepts found in SigPML and the eCl@ss-standard.

Result. Finally, we evaluate the capability of handling large models as they occur in the eCl@ss-standard in form of XML files, which are transformed

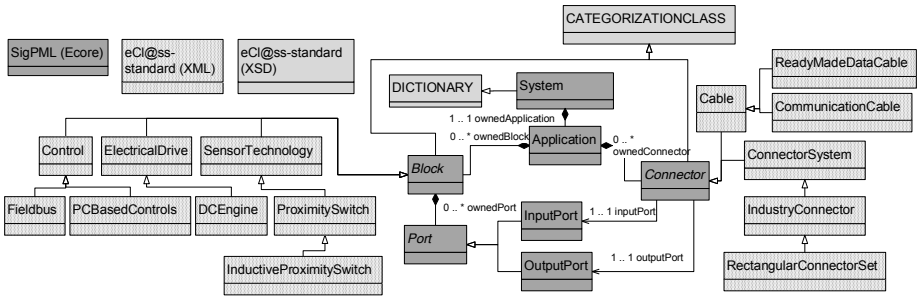

Figure 25: Excerpt of industrial production system meta-model based on SigPML and the eCl@ss-standard.

to XMI files by the XMLIntellEdit framework to enable their use by Extremo. Further, SigPML (only) contains 13 semantic nodes at meta-model level and none at model-level and is thus neglected in Table 4. To summarize, the meta-level contains one *Resource*, i.e., "EPML.ecore", which references 18 different XSD files, that is instantiated by a single XML file, i.e., "eClass9_1_BASIC_EN_SG_27.xmi" (55.6 MB). Moreover, at the model-level there are 487,746 instances of semantic nodes (525 different kinds), 805,097 instances of object properties (500 different kinds), 487,745 instances of data properties (26 different kinds), and 820,356 instances of constraints[9] (88 different kinds).

Consequently, our results indicate that the Extremo-constructed EPML reduces the number of potential candidates for conveyor belt system components, which are available in the eCl@ss standard, by approximately 99.17% (97.05%), i.e., from 487,746 (805,097) to 4,071 (23,752) semantic nodes (object properties) that represent instances (references) of CATEGORIZATIONCLASS and thus potential candidates for instances (references) of (to) Block and Connector in SigPML.

Additional comments. EPML may be extended by either adding further eCl@ss-standard specific concepts, which represent instances of CATEGORIZATIONCLASS, to the meta-model or by expressing the concept of blocks and connectors as concrete (instead of abstract) classes. In more detail, the latter option would move the decision making-process of choosing desired eCl@ss-standard elements from meta-model level to model-level. Although Extremo supports such cases by the means of level-agnostic data handling, we choose to constrain EPML at meta-model level to limit the set of possible types, which can be instantiated at model-level, and thus fit the purpose of modelling conveyor-belt production systems.

[9]Note that the "number of instances" of constraints refers to the number of constraints defined at meta-level and validated at model-level.

	Number of instances	
Common Data Model concept	**Meta-level (types)**	**Model-level (data)**
Resource	1	1
SemanticNode	525	487,746 (4,071)
ObjectProperty	500	805,097 (23,752)
DataProperty	26	487,745
Constraint	88	820,356

Table 4: Instances of imported Common Data Model concepts within the industrial production system modelling case study.

5.2.3. Summary of the demonstration cases

The processes for immigration case study (Section 5.2.1) imports and queries data from different technical spaces, i.e., Ecore meta-models, CSV files, and OWL specifications and the industrial case study (Section 5.2.2) considers XML schemas, XML instances, and Ecore meta-models. Thus, in the first one, we consider a greater variety of technical spaces and smaller models. In contrast, in the second one we address the importing of a lower variety of technical spaces but larger models, i.e., XML instances in the size of multiple gigabytes. Then, in the first case study, we evaluate the ability of EXTREMO in providing assistance during the modeling of resources that are originated from a variety of technical spaces, and in the second case study, we evaluate the applicability of EXTREMO in assistance-scenarios that require dealing with industrially-sized resources.

Table 5 summarizes the number of meta-classes obtained from each assistant in both cases. In the first one, a total of 6 meta-classes were obtained according to the domain-specific concepts, 12 metaclasses were obtained from different ecores and the rest were added by hand. In the second one, a total of 16 meta-classes were obtained from different schemas and descriptions using the XSD assistant and the rest from ecores. Overall, most content in both meta-models was reused from the available resources, which were taken from 4 different technical spaces.

From these demonstration cases, we can answer *RQ1: How useful is* EXTREMO *to solve practical problems in language engineering?* by stating that EXTREMO was helpful in locating elements within heterogeneous resources that helped to create the meta-models. These elements could be directly inserted in the final meta-model. For both cases, most elements in the meta-models were reused from the artefacts in the repository.

5.3. Evaluating format extensibility

In order to evaluate format extensibility, we perform an analytical evaluation of the degree of coverage of the data model of common features found in information modelling approaches [48, 49]. Figures 26 and 27 show a feature

Table 5: Evaluating the usefulness of EXTREMO: results of the experiments

Measures	**Case 1: Processes for immigration**	**Case 2: Production Systems**
Size of metamodel obtained	27 metaclasses	23 metaclasses
From Ecore Assistant	13 metaclasses	7 metaclasses
From Ontologies Assistant	4 metaclasses	N/A
From CSV Assistant	2 metaclasses	N/A
From XSD/XML Assistant	N/A	16 metaclasses
Manually added	8	N/A

diagram (splitted in two parts for readability) displaying these features. Our aim is not to be fully exhaustive, but to cover a reasonable design space for information modelling approaches.

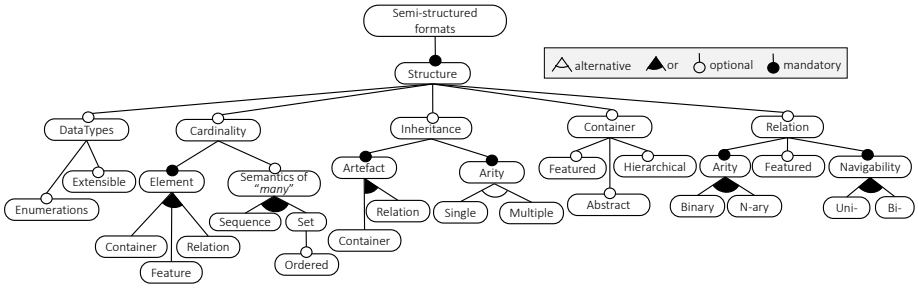

Figure 26: Feature model of characteristics of structured formats (1/2).

Figure 26 shows features related to the space of possible supported structure primitives. Formats to represent semi-structured information are based on some kind of container for data (models, nodes, objects, classes), and on relations among them (features Container and Relation) [49, 50]. Containers may support features (fields, references), may support nesting, and have ways to control their instantiability (e.g., abstractness tag). Relations have an arity, which is typicaly either binary (to model relations between exactly two containers) or n-ary (to model multi-container relations). Similar to containers, some systems may allow relations to own features. Relations may by navigable either in one or both directions [51, 52].

Some systems support some form of inheritance to reuse information [51].

Inheritance relations can normally be set either between containers or relations, and be single (at most one super) or multiple (any number of super elements). Often, systems support a notion of cardinality, to specify the expected range of values of a given element can take. Typically, cardinalities can be attached to containers, relations or features. Some systems permit specifying the semantics of "many" cardinality: a set (no repetition allowed) optionally ordederd, or a

sequence (repetitions allowed) [51, 52]. Finally, many systems may have predefined data types (like integer, String, etc), have support for enumerations, and be extensible with new data types, provided by the user.

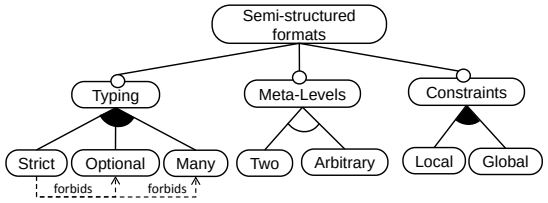

Figure 27: Feature model of characteristics of structured formats (2/2).

Figure 27 captures additional features. Some systems organize their elements in meta-layers, typically two (e.g., models and meta-models), but others support an arbitrary number of them. If meta-levels are supported, some kind of typing is needed between entities in different layers. In some cases, this typing can be optional, or allowed to be multiple. In other cases, the typing is strict, meaning that each element of a certain level is typed exactly by one entity at the level above [53]. This precludes optional typing, multiple typings and typing relations within the same level [54]. Finally, some systems support constraints, typically defined on one meta-level, and evaluated in some meta-level below. These constraints can be local (i.e., involving features of just one entity) or global (if entities of the whole resource need to be access).

Once we have set the design space for information modelling formats, we analyse how different technologies are positioned in it, and the degree of coverage of our data model. A summary of such analysis is shown in Table 6.

Table 6: Features of some information modelling technologies: DataType (E=Extensible, F=Fixed, EN=Enumerations), Cardinality (C=Container, F=Feature, R=Relation, Sem=Configurable semantics of many), Inheritance (C=Container, R=Relation, S=Single, M=Multiple), Container (F=Featured, A=Abstract, H=Hierarchical), Relation (Bin=Binary, N=N-ary, F=Featured, U=Unidirectional, B=Bidirectional), Typing (S=Strict, O=Optional, M=Many), Meta-Levels (2=Two, A=Arbitrary), Constraint (L=Local, G=Global).

System	**DT**	**Card**	**Inh**	**Container**	**Relation**	**Typing**	**Levels**	**Const**
EMF	E, EN	F, Sem	C, M	F, A, H	Bin, U, B	S	2	L, G
UML	E, EN	F, Sem	C, M	F, A, H	N, U, B, F	M	2	L, G
MetaDepth	F, EN	C, F, R, Sem	C, M	F, A, H	Bin, U, B, F	O	A	L, G
XSD	E, EN	F, Sem	C, S	F, H	Bin, U	S	2	L
OWL	E, EN	F, Sem	C, M	F, A	Bin, U, B	O, M	A	L
CSV	F	–	–	F	–	–	–	–
Extremo	F	F	C, M	F, A, H(Rsr)	Bin, U, B	O, M	A	L, G

In the modelling technical space, we have taken three representatives: EMF, UML and MetaDepth. It can be seen that EMF allows enumerations and extensible data types, permits cardinalities on features and fine-tuning the semantics of "many". It supports multiple inheritance on classes, classes (but not references) may have features, and can be abstract. Both packages and classes

can be organized hierarchically (classes may define containment references that contain other classes). Relations are binary, and can be bidirectional (emulated through opposite references). The typing is strict and on a two meta-level architecture. Constraints can be expressed by OCL and can be both local and global. UML offers similar features, but includes N-ary, featured relations (association classes), and typing can be multiple (through overlapping generalization hierarchies). Finally, MetaDepth permits adding cardinalities on nodes, relations (edges) and features. Edges can have features, and there is optional typing on an arbitrary number of meta-levels.

In addition to the modelling technical space, we are interested in evaluating representatives from other technical spaces. First, we selected XSD which offers enumerations and extensible data types as well as cardinalities on features with different configurations for the semantics of "many". Furthermore, XSD offers for element types the following three possibilities: single inheritance, features, and nesting of element types (i.e., hierarchies). XSD only allows binary unidirectional references. Typing in XSD is considered to be strict, except open points in XSD descriptions which allow for any valid XML structure. XSD follows the classical two-level approach and allows for local constraints. For global constraints, additional format languages such as Schematron have to be used. OWL has similar features, but it allows for multiple inheritance between classes which may be also abstract classes. There is no explicit hierarchy based on nesting classes. Relations in OWL may be defined as bi-directional and addition to many other relationship types. Interestingly, OWL allows for optional typing as well as multiple types. Furthermore, arbitrary modeling levels may be defined with OWL, to be more precise, with OWL Full. Local constraints are supported, however, for global constraints, additional constraint languages such as SHACL [55] have to be used.

It can be seen that the data model of Extremo supports most features, with some limitations that can either be overcomed, or are not important for Extremo's goals, as explained next. First, Extremo's data types are currently not extensible. Instead, unsupported data types (e.g., currency) need to be mapped to an existing one (e.g., String) and the can be annotated using MetaData objects. Similarly, enumerations need to be stored as Strings. However, this is not problematic when issuing queries. Cardinality can only be placed on features, and the semantics of "many" is not configurable. However, this is not an important feature to issue queries, and can be reflected using MetaData or Constraint objects. Inheritance is on containers and can be multiple. This is inline with most analyzed systems. Containers can have features and be abstract. However, only resources can be hierarchial. Nonetheless, hierarchy of semantic nodes can be emulated by ObjectProperties. Relations (ObjectProperties) are binary, can be bidirectional (by declaring opposites) and may have features. N-ary or featured associations can be emulated by adding an intermediate SemanticNode. This is the strategy we followed when building the assistant for MetaDepth. Extremo's typing is optional and multiple, supporting an arbitrary number of levels. Finally there is support for both local and global constraints.

Please note that the common data model can also accommodate data formats

with no explicit descriptions, like e.g. CSV. In such a case, each data row would be imported as a semantic node, and each cell as a data property.

Altogether, from this analytical evaluation, we can conclude that most common features of information modelling approaches can be directly mapped to our common data model, or can be emulated. Therefore, we can answer *RQ2: How capable is Extremo to represent information coming from different techno-logical spaces?* by stating that EXTREMO will be able to accommodate most commonly used information modelling approaches.

5.4. Evaluating the integration with external tools

The idea of EXTREMO is to be a modelling assistant easy to integrate in other (meta-)modelling tools. Hence, we have assessed to what extent this integration is feasible, by integrating EXTREMO with a set of tools developed by third-parties. In some cases, the original code was not accessible, while in others it was. In the first case, we used the UML model editor[10] (as shown in Figure 17), the standard Ecore editor and Exeed, an enhanced version of the built-in EMF reflective tree-based editor that enables developers to customize the labels and icons of model elements[11]. All these solutions are based on a TreeEditor, an extensible part of the Eclipse infraestructure. Since a drag and drop integration is not possible because of restrictions to access to the original code, the solution was performed by means of the action extension point. Each of these integrations costed 234 lines of Java code (LOCs) in average, which can be considered as very light.

In the second case, we used DSL-tao, which was built by our team. In this case, the code was available, and performing a solution by means of the drag and drop extension point. costed 134 lines of Java code (LOC).

Table 7 shows details on the number of LOCs for each integration. The integration mechanisms by means of actions and drag and drop are already provided by the tool (marked with an asterisk) and do not need to be provided by the developer. Therefore, most of the code needed was related to the transformation from the instances of our data model (Figure 2) to the classes of the modelling tool (cf. Figures 16 and 18). In the case of the integration made by means of actions, the method execute needs to resolve the editor part that will receive the portion of the model instance and the selected elements from the views before to create the new elements of the transformation (shown in row 3). The necessary LOCs to transform nodes, data and object properties are detailed in rows 4-6 of the table.

Thus, from this study, we can answer *RQ3: How integrable is* EXTREMO*?* by stating that integration of EXTREMO is lightweight for modelling tools based on tree or GEF-based editors.

[10] UML2-MDT, `www.eclipse.org/modeling/mdt`

[11] Epsilon Exeed, `http://www.eclipse.org/epsilon/`

	DSL-tao	**EcoreEditor**	**UML2Editor**	**ExeedEditor**
Ext. Point Used	drop	actions	actions	actions
Ext. Point Integration	59*	49*	49*	49*
Tree Selection Solver	-	163	165	163
SemanticNode	24	8	4	8
DataProperty	27	33	4	33
ObjectProperty	24	9	32	9

Table 7: LOCs for integrating EXTREMO with other tools

5.5. Discussion and threats to validity

As we have seen in the three preceding subsections, we were able to use EXTREMO to help in constructing DSLs by reusing heterogeneous artefacts (some of which had large size); we analysed the degree in which the data model of EXTREMO is able to accommodate possible information modelling approaches; and how easy is it to integrate EXTREMO with external (meta-)modelling tools. While the results are positive, there are of course also potential threats to the validity of the experiments. According to Wohlin et al. [56], there are four basic types of validity threats that can affect the validity of our study. We cover each of these in the following paragraphs.

5.5.1. Construct Validity

Construct validity is concerned with the relationship between theory and what is observed. The demonstration cases in Section 5.2 focussed on evaluating the use of EXTREMO with assistants for different technologies, and standards (like eCl@ss) developed by third parties. However, although taking realistic requirements, the DSLs to be constructed were devised by us. Therefore, further studies would need to be performed by constructing DSLs with requirements specified by third parties.

The evaluation of the demonstration cases focussed on DSL construction. However, it used artefacts acting as descriptors (XSD) and at the model/data level (XML documents). While this shows that EXTREMO can be used to extract information at the model level, a further study would be needed to assess the usefulness of EXTREMO for domain-specific modelling. However, please note that creating a meta-model is a structural modelling activity already.

5.5.2. Conclusion Validity

Conclusion validity is concerned with the relationship between the treatment and the outcome. We considered two demonstration cases from different domains, seven format languages from four technical spaces, and integrated our approach with four modeling editors. While these numbers may be not enough to reason about statistical relevance, they still show a clear tendency of the usefulness, applicability, and integrability of our approach.

5.5.3. Internal Validity

Internal validity checks whether the treatment used in the experiment actually causes the outcome. We were the performers of both demonstration cases. While the performer of one of the case study was not involved in the development of Extremo, a user study would be needed to further assess the tool usability and the subjective usefulness of Extremo. However reporting on a user study would deserve a separate publication, and we will tackle this issue in future work. Similarly, the integration of Extremo with external tools was also performed by us. Although lightweight in terms of LOC, it could be more demanding for other developers in terms of effort.

Another aspect is that we set the class as the unit of reuse, neglecting properties. We believe this is a good indicator as the number of classes outperforms that of properties.

Having good resources available is crucial for the approach to work properly. We did not evaluate how easy is it to perform this phase of resource collection (since this phase is out of the scope of our tool), but we evaluated how large was the contect of the repository, though.

5.5.4. External Validity

Regarding external validity (i.e., generalizability of the results), we did not include an explicit evaluation of query extensibility, because the extension points we have defined permit adding new queries by using arbitrary Java code. Table 9 in the appendix lists a collection of queries we have defined by implementing the extension point and that covers a set of accepted quality criteria in conceptual modelling [24].

For RQ3 (integrability) we did not evaluate the integration of Extremo with text-based modelling tools, e.g., built with Xtext, but we have assessed to what extent this integration is feasible, by integrating Extremo with a set of tools developed by third-parties (and also developed by us). In addition, we integrated Extremo with other tools within Eclipse, but not with tools in other IDEs, like JetBrains. While Extremo is an Eclipse plugin, its Core subsystem (described in Section 4.1) is largely independent from it (in contrast to the UI subsystem 4.2). Hence, migrating the Core into JetBrains would require little effort, but the UI subsystem (dealing with visualization and interaction with resources and query results) would need to be redesigned.

We did not present a formal evaluation of scalability or performance, which are left for future work. Regarding the former, the XML artefacts considered in the second demonstration case reached a size of 55Mb. Moreover, resources are imported and persisted using NeoEMF, a model persistence solution designed to store models in NoSQL datastores, which is able to handle very large models efficiently (e.g., models of more than 40.000.000 elements were reported to be created in [57]).

Regarding performance, our experience and preliminary evaluations indicate that resource import time is linear in the size of the resource. Typically, it takes a few seconds for resources of sizes in the order of hundreds of elements. While we plan to optimize this performance, this is a one-time operation, and once

a resource is imported, it can be handled through NeoEMF. Regarding query performance, those that need to traverse the whole resource are in the order of one second for sizes up to thousands of elements. However, they may become a bottleneck for larger resources. To alleviate this issue, we cache both the input parameters for predicate-based searches (init operation shown in Figure 6) and the query results, while NeoEMF lazy-loading mechanisms that transparently brings into memory model elements only when they are accessed. Further optimizations to speed-up queries, e.g., based on the creation of derived features and indexes for the resources [58], are left for future work.

6. Related work

The increasing complexity of software development has prompted the need for code recommenders for example, for API usage, or program quick fix. However, although code recommenders are increasingly used in programming IDEs [7, 8], there is lack of such assistive tools for (meta-)modelling in MDE.

The closest work to our proposal is [59, 60, 61], where a generic architecture for model recommenders is proposed. The architecture is extensible, in the sense that different recommender strategies can be plugged-in. In contrast, the extensibility of our approach is in the supported data source, while we specifically focus on the extraction of knowledge from these sources. In addition, our approach supports out-of-the-box visualization and extensible query facilities.

Other approaches to model recommendation focus on proposing suitable ways to complete a model with respect to a meta-model [62]. Hence, using constraint solving techniques, the system proposed ways to complete a model so that it becomes a valid instance of a meta-model. In [63] the authors use ontologies in combination with domain-specific modelling, and hence can use ontology reasoners to provide reasoning services like model validation, inconsistency explanation, and services to help in the model construction phase.

Some approaches propose a common architecture to index and represent models, with the purpose of reuse. For example, in [64] the authors transform SysML models into a "universal" representation model called RHSP, which can be queried to discover reusable models. They support queries based on partial models (and model similarity search) and natural language (similar to our synonym searchs). In our case the queries are extensible, and our data model provides richer support for representing model features, including constraints.

Instead of using a common data model, an alternative design would have been to use model adapters, in the style of the Epsilon model management languages [65]. In this approach, the languages do not access the particular information technology (EMF, XML) directly, but through a model connectivity layer. This is an interface that can be implemented to enable uniform access to different technologies. We opted for a common data model, where the different heterogeneous elements are reified uniformly, and stored using NeoEMF, hence providing scalability and performance.

Storing artefacts in a database, to enable their flexible query has also been applied to source code [15, 66]. In our case, the artefacts come from different

heterogeneous sources, and hence we need to transform them into the common data model. Our query approach is extensible, based on extension points.

Other works have considered the exchange of models/data between different meta-modelling tools [67] or technical spaces [68]. In [67] the authors propose a solution that creates transformation between different meta-modelling technologies by means of declarative mappings. Our approach differs from these works by mapping the technical spaces into a common data model instead of establishing mappings between individual technical spaces. As a result, our approach is independent of a single technical space and thus enables the import, persistence, and querying of interdependent concepts that are originated from distinctive technical spaces and may be lost within single mappings. In [68] the maintenance of intra-space transformations is improved by automating the discovery and reuse of mappings between schema elements. In contrast, Extremo provides assistance during the import of artifacts from different technical spaces and the creation of new languages and models that are based on existing technical spaces, such as Ecore, regardless of their originating technical space. Although Extremo requires to specify assistants for different technical spaces, existing EMF-based work that bridges technical spaces, such as XMLText [42] for XML schema, can be reused and (only) requires the specification of a mapping within the same technical space, i.e., Ecore in case of Extremo and XMLText.

Some researchers have exploited ontologies for creating DSLs [69]. For example, in [70] the authors advocate the use of (OWL) ontologies in the domain analysis phase of DSL construction. As they target textual DSLs, they propose a tool for the automated generation of a textual grammar for the DSL. In a similar vein, in [71], the authors generate meta-model design templates from OWL ontologies, which are later manually refined into domain meta-models. In our approach, we assume that not all the required information to create a meta-model is present in one ontology, but typically such information is scattered in informational resources of different kinds, like ontologies, RDF data, or meta-models.

Combining modeling approaches from MDE with ontologies has been studied in the last decade [72]. There are several approaches to transform Ecore-based models to OWL and back, e.g., cf. [73, 74]. In addition, there exist approaches that allow for the definition of ontologies in software modeling languages such as UML by using dedicated profiles [75]. Moreover, there are approaches which combine the benefits of models and ontologies such as done in [76, 77] for reasoning tasks. Not only the purely structural part of UML is considered, but some works also target the translations of constraints between these two technical spaces by using an intermediate format [78]. For the data import, we may build on these mentioned approaches, but we focus on recommendation services exploiting the imported data from different technical spaces to build domain-specific modeling languages.

Finally, there are some approaches directed to search relevant models within a repository. Their aim is slightly different from our goal, which is looking for relevant information within a repository. Moogle [34] is based on textual, "Google-

like" queries, similar to ours. As they focus on EMF model-level queries, they use the meta-model information for filtering, like we do as well. However, our queries are extensible, and hence new types of queries can be defined. Moreover, their results are shown in textual format and we parse and aggregate the results as well as offer graphical visualization. EMF query is directed to search EMF models [79], using OCL queries or text-based search. The latter may include regular expressions, but does not look for relevant synonyms as we do. Moreover, our extensible approach supports technologies like Ecore, OWL and RDF. Furthermore, there are dedicated approaches offering search capabilities tailored for a specific modelling domain such as [80, 81]. Although these approaches allow to reason on behavioral similarity aspects, we aim for general model search support independently of the modelling domain and technical space.

Work	Assistance	Heterogeneous Sources	Common Model	Queries
Dyck et al. [59, 60, 61]	✓	✗	✗	✗
Sen et al. [62]	✓	✗	✗	✗
Walter et al. [63]	✓	∼ (OWL)	✗	✗
Mendieta et al. [64]	✗	∼ (SysML)	✓	Not extensible
Kern, Dimitrieski et al. [67, 68]	✗	✓	✗	✗
Ontology-based DSL development [70, 71]	✗	∼ (OWL)	✗	✗
Moogle [34]	✗	∼ (EMF)	✗	Not extensible
Extremo	✓	✓	✓	Extensible

Table 8: Summary comparison of Extremo and closest related works

Table 8 presents a feature-based summary of Extremo and the closest related works. In summary, our approach is novel in that it provides an assistant system that profits from the integration and querying of heterogeneous information sources. Although some approaches have focussed on using specific technologies, such as Ontologies [63, 69, 70, 71], to build (meta-)models, our approach is more general as a result of supporting different technologies. Moreover, there exist approaches that establish bridges between technical spaces [67, 68], our contribution differs by providing a common data model to store, query, and establish assistance for information from different technical spaces. Further, in contrast to other existing approaches, which have devised query mechanisms to search for relevant models in a repository [34], our querying mechanism is extensible. Finally, some approaches to provide model assistance are based on model completion (w.r.t. a meta-model) [62] or provide a generic mechanism to plug-in assistants [59, 60, 61]. Contrarily, we contribute a specific architecture to support assistance that is based on querying heterogeneous information sources.

7. Conclusions and future work

In this paper, we have presented Extremo, an extensible assistant for modelling and meta-modelling. The system is able to gather information from different technological spaces (like ontologies, RDF, XML or EMF), by representing this information under a common data scheme. This enables their uniform querying and visualization. Extremo is independent of the particular modelling tool, but easily integrated with them due to its modular architecture

based on extension points. We have shown its integration with DSL-tao and several other tools, and used it for the construction of DSLs in the e-Government and production systems domain. We have performed an evaluation of several aspects, showing good results.

In the future, we plan to connect Extremo with meta-model repositories, such as MDEForge [82]. Extremo currently supports a re-active integration mode, where the assistant is explicitly invoked.

Similar to [60], we would also like to explore pro-active modes for assistance. For this purpose, we plan to use recommendation techniques based on rich contextual models, which take into account not only the current model state, but also the user interaction with the IDE [83]. We are currently considering a user study, made of two parts. First, we will evaluate the perceived usefulness of Extremo by engineers in order to perform different modelling tasks (e.g., construct or modify a model). Second, we will compare the quality of the resulting models, and the effectivenes of the modelling task, with respect to not using assistance. Finally, we plan to improve query efficiency by using model indexes [58].

Acknowledgements. We would like to thank to the reviewers for their valuable comments. This work was supported by the Ministry of Education of Spain (FPU grant FPU13/02698); the Spanish MINECO (TIN2014-52129-R); the R&D programme of the Madrid Region (S2013/ICE-3006); the Austrian agency for international mobility and cooperation in education, science and research (OeAD) by funds from the Austrian Federal Ministry of Science, Research and Economy - BMWFW (ICM-2016-04969); and by the Christian Doppler Forschungsgesellschaft, the Federal Ministry of Economy, Family and Youth and the National Foundation for Research, Technology and Development, Austria.

References

[1] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, 2nd Edition, Morgan & Claypool, 2017.

[2] D. C. Schmidt, Guest editor's introduction: Model-driven engineering, Computer 39 (2) (2006) 25–31.

[3] A. R. da Silva, Model-driven engineering: A survey supported by the unified conceptual model, Computer Languages, Systems & Structures 43 (Supplement C) (2015) 139 – 155. doi:https://doi.org/10.1016/j.cl.2015.06.001.

[4] UML 2.5 OMG specification, http://www.omg.org/spec/UML/2.5/.

[5] S. Kelly, J. Tolvanen, Domain-Specific Modeling - Enabling Full Code Generation, Wiley, 2008.

[6] J. E. Hutchinson, J. Whittle, M. Rouncefield, Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure, Sci. Comput. Program. 89 (2014) 144–161.

[7] Eclipse Code Recommenders, http://www.eclipse.org/recommenders.

[8] M. P. Robillard, R. J. Walker, T. Zimmermann, Recommendation systems for software engineering, IEEE Software 27 (4) (2010) 80–86.

[9] J. Bézivin, Model driven engineering: An emerging technical space, in: Generative and Transformational Techniques in Software Engineering, International Summer School, GTTSE, Vol. 4143 of Lecture Notes in Computer Science, Springer, 2005, pp. 36–64.

[10] I. Kurtev, J. Bézivin, M. Aksit, Technological spaces: An initial appraisal, in: International Symposium on Distributed Objects and Applications, DOA 2002, 2002.
URL http://doc.utwente.nl/55814/

[11] Á. M. Segura, A. Pescador, J. de Lara, M. Wimmer, An extensible meta-modelling assistant, in: IEEE EDOC, IEEE Computer Society, 2016, pp. 1–10.

[12] J. de Lara, E. Guerra, Deep meta-modelling with metadepth, in: TOOLS, Vol. 6141 of Lecture Notes in Computer Science, Springer, 2010, pp. 1–20.

[13] eCl@ss Standard 9.0, http://wiki.eclass.eu/.

[14] P. A. Ménard, S. Ratté, Concept extraction from business documents for software engineering projects, Autom. Softw. Eng. 23 (4) (2016) 649–686.

[15] E. Linstead, S. K. Bajracharya, T. C. Ngo, P. Rigor, C. V. Lopes, P. Baldi, Sourcerer: mining and searching internet-scale software repositories, Data Min. Knowl. Discov. 18 (2) (2009) 300–336.

[16] S. Subramanian, L. Inozemtseva, R. Holmes, Live API documentation, in: ICSE '14, ACM, 2014, pp. 643–652.

[17] C. Treude, M. P. Robillard, Augmenting api documentation with insights from stack overflow, in: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, ACM, New York, NY, USA, 2016, pp. 392–403.

[18] F. Basciani, J. D. Rocco, D. D. Ruscio, L. Iovino, A. Pierantonio, Automated clustering of metamodel repositories, in: CAiSE, Vol. 9694 of Lecture Notes in Computer Science, Springer, 2016, pp. 342–358.

[19] A. Pescador, A. Garmendia, E. Guerra, J. S. Cuadrado, J. de Lara, Pattern-based development of domain-specific modelling languages, in: MoDELS, 2015, pp. 166–175.

[20] K. Czarnecki, M. Antkiewicz, Mapping features to models: A template approach based on superimposed variants, in: GPCE, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 422–437.

[21] A. Polzer, D. Merschen, G. Botterweck, A. Pleuss, J. Thomas, B. Hedenetz, S. Kowalewski, Managing complexity and variability of a model-based embedded software product line, Innovations in Systems and Software Engineering 8 (1) (2012) 35–49.

[22] J. S. Cuadrado, E. Guerra, J. de Lara, A component model for model transformations, IEEE Transactions on Software Engineering 40 (11) (2014) 1042–1060.

[23] N. Moha, Y. Guéhéneuc, L. Duchien, A. L. Meur, DECOR: A method for the specification and detection of code and design smells, IEEE Trans. Software Eng. 36 (1) (2010) 20–36.

[24] D. Aguilera, C. Gómez, A. Olivé, Enforcement of conceptual schema quality issues in current integrated development environments, in: Proc. CAiSE, Vol. 7908 of Lecture Notes in Computer Science, Springer, 2013, pp. 626–640.

[25] I. Kurtev, M. Aksit, J. Bézivin, Technical Spaces: An Initial Appraisal, in: Proc. of CoopIS, 2002.

[26] C. Atkinson, T. Kühne, Reducing accidental complexity in domain models, Software and System Modeling 7 (3) (2008) 345–359.

[27] C. Atkinson, B. Kennel, B. Goß, The Level-Agnostic Modeling Language, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 266–275.

[28] J. de Lara, E. Guerra, *A Posteriori* typing for model-driven engineering: Concepts, analysis, and applications, ACM Trans. Softw. Eng. Methodol. 25 (4) (2017) 31:1–31:60.

[29] OMG, SMOF 1.0, `http://www.omg.org/spec/SMOF/1.0/` (2013).

[30] Z. Diskin, S. Kokaly, T. Maibaum, Mapping-aware megamodeling: Design patterns and laws, in: Proc. SLE, Vol. 8225 of Lecture Notes in Computer Science, Springer, 2013, pp. 322–343.

[31] J. de Lara, E. Guerra, R. Cobos, J. Moreno-Llorena, Extending deep metamodelling for practical model-driven engineering, Comput. J. 57 (1) (2014) 36–58.

[32] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework, Addison-Wesley, 2008.

[33] OMG, MOF 2.5.1, `http://www.omg.org/spec/MOF/2.5.1/` (2016).

[34] D. Lucrédio, R. P. de Mattos Fortes, J. Whittle, MOOGLE: a metamodel-based model search engine, Software and System Modeling 11 (2) (2012) 183–208.

[35] OCL 2.4. specification, `http://www.omg.org/spec/OCL/`.

[36] M. F. Porter, An algorithm for suffix stripping, Program 40 (3) (2006) 211–218.

[37] M. Lesk, Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone, in: SIGDOC, ACM, 1986, pp. 24–26.

[38] G. A. Miller, Wordnet: A lexical database for english, Comm. ACM 38 (11) (1995) 39–41.

[39] E. Kharlamov, B. C. Grau, E. Jiménez-Ruiz, S. Lamparter, G. Mehdi, M. Ringsquandl, Y. Nenov, S. Grimm, M. Roshchin, I. Horrocks, Capturing industrial information models with ontologies and constraints, in: Proc ISWC, Part II, Vol. 9982 of Lecture Notes in Computer Science, 2016, pp. 325–343.

[40] B. Motik, I. Horrocks, U. Sattler, Adding integrity constraints to OWL, in: Proc. OWLED, Vol. 258 of CEUR Workshop Proceedings, CEUR-WS.org, 2007.
URL `http://ceur-ws.org/Vol-258`

[41] A. Benelallam, A. Gómez, G. Sunyé, M. Tisi, D. Launay, Neo4emf, A scalable persistence layer for EMF models, in: Proc. ECMFA, Vol. 8569 of Lecture Notes in Computer Science, Springer, 2014, pp. 230–241.

[42] P. Neubauer, A. Bergmayr, T. Mayerhofer, J. Troya, M. Wimmer, XMLText: From XML Schema to Xtext, in: Proceedings of SLE, 2015, pp. 71–76.

[43] P. Neubauer, R. Bill, T. Mayerhofer, M. Wimmer, Automated generation of consistency-achieving model editors, in: IEEE SANER, 2017, pp. 127–137.

[44] P. Neubauer, R. Bill, M. Wimmer, Modernizing domain-specific languages with xmltext and intelledit, in: IEEE SANER, 2017, pp. 565–566.

[45] Eclipse Graphical Editing Framework, `https://eclipse.org/gef/`.

[46] P. Runeson, M. Host, A. Rainer, B. Regnell, Case Study Research in Software Engineering: Guidelines and Examples, 1st Edition, Wiley Publishing, 2012.

[47] OMG Business Process Model and Notation, `http://www.bpmn.org/`.

[48] J. Mylopoulos, Characterizing Information Modeling Techniques, Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 17–57.

[49] H. Kern, A. Hummel, S. Kühne, Towards a comparative analysis of meta-metamodels, in: Proceedings of the compilation of the co-located workshops, DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, and VMIL'11, Portland, OR, USA, October 22 - 27, 2011, 2011, pp. 7–12.

[50] P. Buneman, Semistructured data, in: Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM Press, 1997, pp. 117–121.

[51] A. Olivé, Conceptual modeling of information systems, Springer, 2007. doi:10.1007/978-3-540-39390-0.

[52] S. Abiteboul, P. Buneman, D. Suciu, Data on the Web: From Relations to Semistructured Data and XML, Morgan Kaufmann, 1999.

[53] C. Atkinson, T. Kühne, Profiles in a strict metamodeling framework, Sci. Comput. Program. 44 (1) (2002) 5–22.

[54] T. Kühne, Matters of (meta-)modeling, Software and System Modeling 5 (4) (2006) 369–385.

[55] Shapes Constraint Language (SHACL), https://w3c.github.io/data-shapes/shacl/.

[56] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, Experimentation in Software Engineering, Springer, 2012.

[57] G. Daniel, G. Sunyé, A. Benelallam, M. Tisi, Improving memory efficiency for processing large-scale models, in: BigMDE, University of York, York, UK, United Kingdom, 2014.
URL https://hal.inria.fr/hal-01033188

[58] K. Barmpis, D. S. Kolovos, Towards scalable querying of large-scale models, in: ECMFA, Vol. 8569 of Lecture Notes in Computer Science, Springer, 2014, pp. 35–50.

[59] A. Dyck, A. Ganser, H. Lichter, Enabling model recommenders for command-enabled editors, in: MDEBE, 2013, pp. 12–21.

[60] A. Dyck, A. Ganser, H. Lichter, A framework for model recommenders - requirements, architecture and tool support, in: MODELSWARD, 2014, pp. 282–290.

[61] A. Dyck, A. Ganser, H. Lichter, On designing recommenders for graphical domain modeling environments, in: MODELSWARD, 2014, pp. 291–299.

[62] S. Sen, B. Baudry, H. Vangheluwe, Towards domain-specific model editors with automatic model completion, Simulation 86 (2) (2010) 109–126.

[63] T. Walter, F. S. Parreiras, S. Staab, An ontology-based framework for domain-specific modeling, Software and Systems Modeling 13 (1) (2014) 83–108.

[64] R. Mendieta, J. L. de la Vara, J. Llorens, J. Álvarez-Rodríguez, Towards effective sysml model reuse, in: Proc. MODELSWARD, SCITEPRESS, 2017, pp. 536–541.

[65] R. F. Paige, D. S. Kolovos, L. M. Rose, N. Drivalos, F. A. C. Polack, The design of a conceptual framework and technical infrastructure for model management language engineering, in: ICECCS, IEEE Computer Society, 2009, pp. 162–171.

[66] E. Hajiyev, M. Verbaere, O. de Moor, K. D. Volder, Codequest: querying source code with datalog, in: Proc. OOPSLA 2005, ACM, 2005, pp. 102–103.

[67] H. Kern, F. Stefan, V. Dimitrieski, M. Čeliković, Mapping-based exchange of models between meta-modeling tools, in: DSM, ACM, New York, NY, USA, 2014, pp. 29–34.

[68] V. Dimitrieski, M. Čeliković, N. Igić, H. Kern, F. Stefan, Reuse of rules in a mapping-based integration tool, in: SoMet, Springer International Publishing, Cham, 2015, pp. 269–281.

[69] R. Tairas, M. Mernik, J. Gray, Using ontologies in the domain analysis of domain-specific languages, in: TWOMDE, 2008, pp. 20–31.

[70] I. Ceh, M. Crepinsek, T. Kosar, M. Mernik, Ontology driven development of domain-specific languages, Comput. Sci. Inf. Syst. 8 (2) (2011) 317–342.

[71] A. Ojamaa, H. Haav, J. Penjam, Semi-automated generation of DSL meta models from formal domain ontologies, in: MEDI, 2015, pp. 3–15.

[72] D. Gasevic, D. Djuric, V. Devedzic, Model Driven Engineering and Ontology Development, Springer, 2009.

[73] T. Walter, F. S. Parreiras, G. Gröner, C. Wende, OWLizing: Transforming Software Models to Ontologies, in: ODiSE, 2010, pp. 7:1–7:6.

[74] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, M. Wimmer, Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages, in: MoDELS, 2006, pp. 528–542.

[75] M. Milanovic, D. Gasevic, A. Giurca, G. Wagner, V. Devedzic, Towards Sharing Rules Between OWL/SWRL and UML/OCL, ECEASST 5.

[76] F. S. Parreiras, S. Staab, A. Winter, On marrying ontological and metamodeling technical spaces, in: FSE, 2007, pp. 439–448.

[77] F. S. Parreiras, S. Staab, Using ontologies with UML class-based modeling: The TwoUse approach, DKE 69 (11) (2010) 1194–1207.

[78] D. Djuric, D. Gasevic, V. Devedzic, V. Damjanovic, A UML Profile for OWL Ontologies, in: MDAFA, 2004, pp. 204–219.

[79] EMF Query, https://projects.eclipse.org/projects/modeling.emf.query.

[80] B. Bislimovska, A. Bozzon, M. Brambilla, P. Fraternali, Textual and content-based search in repositories of web application models, TWEB 8 (2) (2014) 1–11.

[81] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, J. Mendling, Similarity of business process models: Metrics and evaluation, Inf. Syst. 36 (2) (2011) 498–516.

[82] J. D. Rocco, D. D. Ruscio, L. Iovino, A. Pierantonio, Collaborative repositories in model-driven engineering, IEEE Software 32 (3) (2015) 28–34.

[83] M. Gasparic, G. C. Murphy, F. Ricci, A context model for ide-based recommendation systems, Journal of Systems and Software 128 (2017) 200–219.

Appendix: List of queries

Table 9 shows the list of pre-defined queries of EXTREMO, which were defined by implementing the provided extension points. The user may provide additional queries by implementing the extension point (in Java). Some of these queries come from catalogues of accepted quality criteria in conceptual modelling [24]. We divide them into predicate and custom, and depict the element they inspect (filterBy).

SearchConfiguration	filterBy	Description
Predicate Based Search		
A NamedElement has a name	NamedElement	Checks if a NamedElement object has a name that matches with a value. Options: name: PrimitiveTypeParam typed as string
All instances of a NamedElement	NamedElement	Returns all the NamedElements that are instances of another one. Options: type: ModelTypeParam and recursive: PrimitiveTypeParam typed as boolean
A node with a property with value X	SemanticNode	Checks if a SemanticNode object has a data property which has a concrete value. Options: value: PrimitiveTypeParam typed as string
A node has more than a number of parents	SemanticNode	Checks if a SemanticNode object has more than a number of supers instances. Options: parents: PrimitiveTypeParam typed as int
A node has more than a number of children	SemanticNode	Checks if a SemanticNode object has more than a number of subs instances. Options: children: PrimitiveTypeParam typed as int
Attributes are overloaded	SemanticNode	Checks if a SemanticNode object contains more than a number of DataProperties. Options: maxattrs: PrimitiveTypeParam typed as int
References are overloaded	SemanticNode	Checks if a n: SemanticNode object contains more than a number of ObjectProperties. Options: maxrefs: PrimitiveTypeParam typed as int
An abstract node without children	SemanticNode	Checks if a SemanticNode object is abstract and there are no supers instances.
An abstract node with an unique child	SemanticNode	Checks if a SemanticNode object is abstract and it has only a supers instance.
Data Properties Value	DataProperty	Checks if a DataProperty object has a concrete value. Options: valuefield: PrimitiveTypeParam typed as string
Data Properties Value Range	DataProperty	Checks if the integer value of a DataProperty object ranges between a minimum and a maximum. Options: minvaluefield: PrimitiveTypeParam typed as int and maxvaluefield: PrimitiveTypeParam typed as int
Custom Search		
Nodes without descriptions	Resource	For a resource, split the nodes in two groups. The first one refers to the nodes with descriptions are the second one refers to the nodes without descriptions. Options: resource: ModelTypeParam typed as Resource
Isolated nodes	Resource	Checks if a resource contains nodes that are isolated. Options: resource: ModelTypeParam typed as Resource
A hierarchy is too deep	Resource	Checks if a node is too deep on a level of hierarchy. Options: maxdepth: PrimitiveTypeParam typed as int
No node is referred from too many others	Resource	In a resource, checks if there is a node that is referred from too many others. Options: maxrefs: PrimitiveTypeParam typed as int and resource: ModelTypeParam typed as Resource
Hierarchy Splitter	Resource	For a resource, split the nodes in groups. In every group a inheritance hierarchy is left. Options: resource: ModelTypeParam typed as Resource

Table 9: List of simple search configurations (queries)