Towards Enabling Mobile Domain-specific Modelling

Diego Vaquero-Melchor, Antonio Garmendia, Esther Guerra and Juan de Lara
Computer Science Department, Universidad Autónoma de Madrid, Madrid, Spain

Keywords: Domain-specific Modelling, Flexible Modelling, Mobile Applications.

Abstract: Model-Driven Engineering (MDE) promotes an active use of models in all phases of software development. In this paradigm, the design and usage of Domain-Specific Languages (DSL) for modelling in a certain application area is frequent. While in MDE, modelling has been traditionally supported by desktop computers, in this position paper we analyse useful scenarios for modelling using mobile devices, like smartphones or tablets. Moreover, we present a working architecture and a prototype tool, called *DSL-comet*, which enable collaborative mobile modelling and integrate seamlessly desktop and mobile graphical modelling environments.

1 INTRODUCTION

Model-Driven Engineering (MDE) (Brambilla et al., 2012) promotes the active use of models in all phases of software development. Hence, models are used to specify, analyse, simulate, test, execute and generate code for the final applications. While it is possible to define those models using general-purpose languages like the UML, their construction using Domain-Specific Languages (DSLs) tailored to particular domains is frequent (Kelly and Tolvanen, 2008).

A primary goal of models in MDE is to serve as an automation mechanism for development tasks like code generation. Thus, even though if initial phases of modelling may take place in informal settings like whiteboards or using pen and paper, models need to become precisely defined to be machine processable. Traditionally, modelling in MDE takes place in desktop computers assisted by modelling tools, like those based in Eclipse/EMF (Steinberg et al., 2008). While this is useful for late phases of model development, it introduces rigidity and prevents using modelling and models in flexible scenarios that imply mobility, collaboration and make use of contextual information.

In this position paper, we claim that modelling using DSLs is useful in scenarios requiring mobility, collaboration and context. We analyse several mobile modelling situations of interest, and present an architecture and prototype tool for the discussed scenarios. In particular, our approach permits the automatic generation of both desktop and mobile graphical modelling environments from a single description, as well as the seamless editing of models in both kinds of environments. Communication between desktop and mobile environments is made through a dedicated server. Our desktop environment is an Eclipse plugin based on Sirius (Sirius, 2016) as a platform for graphical modelling. The mobile environment is based on iOS, and permits model sharing and local collaborative model editing via local ad-hoc WiFi networks.

The paper is organized as follows. Section 2 motivates the need for mobile domain-specific modelling, describing scenarios of interest. Section 3 describes our proposed architecture. Section 4 introduces *DSL-comet*, our prototype tool. Section 5 compares our approach and tool with related works. Finally, Section 6 draws some conclusions and lines for future work.

2 SCENARIOS FOR MOBILE MODELLING

In this section, we discuss some scenarios where mobile domain-specific modelling is useful.

2.1 Combined Modelling

In the first scenario, a model can be defined in a desktop environment and then it can be used in a mobile context, or vice versa. Fig. 1 shows a schema of this situation, where a server stores the models. Those models can be downloaded for their editing in mobile and desktop modelling environments indistinctly.

Some applications of this scenario include modelling in factories or remote locations (e.g., a wind

Vaquero-Melchor, D., Garmendia, A., Guerra, E. and Lara, J.
Towards Enabling Mobile Domain-specific Modelling.
DOI: 10.5220/0006002501170122
In *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 2: ICSOFT-PT*, pages 117-122
ISBN: 978-989-758-194-6

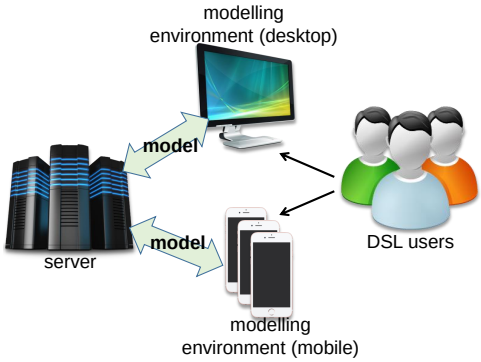


Figure 1: Combined desktop/mobile modelling.

turbine) through mobile devices. For example, an operator of a wind turbine may need to inspect a model of the turbine controller on site, and maybe change its parameters according to the current atmospheric situation.

The seamless integration of desktop and mobile modelling also enables informal, agile meetings between engineers, who may use a combination of tablets and desktop monitors for model visualization. Finally, this scenario also has utility in education. In this setting, professors may create models or modelling exercises in the desktop computer. These exercises or modelling lessons can then be used by students for learning in mobility. The solution to the exercises could be uploaded by the students to the server, and be graded by the professor in the desktop.

2.2 Mobile Collaborative Modelling

In this scenario, one may benefit from the short-range communication capabilities of mobile devices to enable local collaboration (e.g., for joint model construction). Hence, there is no need to use a remote server to orchestrate and coordinate the collaboration, which may incur in delays or can be impossible in remote locations where no data connection is available. Instead, collaboration can take place by using the short-range communication capabilities of mobile devices like Bluetooth or WiFi.

Fig. 2 depicts this scenario. First, one user downloads from the server a palette with the elements that can appear in the model. Then, this user sets a local WiFi network, and invites other nearby users to the collaborative session. The collaboration rules may be customizable depending on the particular application (e.g., token-based, with either implicit or explicit assignment of the modification token). When the session finishes, the model can be stored in the server or locally.

Applications of this scenario include those in Section 2.1, but enhanced with local collaboration facilities.

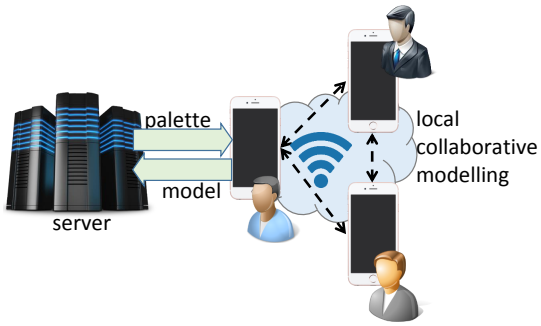


Figure 2: Local collaborative modelling.

2.3 Context-based Modelling

Mobile devices can access contextual information. This can be useful for modelling scenarios (Bettini et al., 2010). For example, we may present different parts of the model, or allow different editing actions, depending on the context (see Fig. 3). Context may include information about the device state (e.g., battery, size of screen) or external information (e.g., position, time, weather conditions).

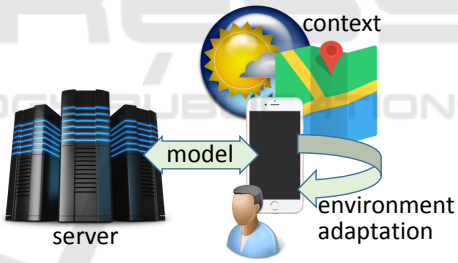


Figure 3: Mobile contextual modelling.

An application of this scenario is domotics. For instance, a mobile app may present a model of the devices in the room where the house owner is currently located, updating this view when the owner moves to another room. By manipulating the model, the owner may change the operating conditions of the devices.

From an analysis of the different scenarios, we identify the following requirements for a modelling platform. (1) In order to allow combined modelling, it should be easy to generate both desktop and mobile environments. (2) It should enable the seamless use of models in desktop and mobile devices. (3) Model visualization in the mobile environment should be adapted to the reduced screen size. (4) Model editing in the mobile environment should be adapted to support typical mobile interaction gestures (e.g., swipe, tap and pinch). (5) In order to allow the second sce-

nario, it should support local collaboration in the mobile environment. (6) In order to allow the third scenario, it should enable context adaptation in the mobile environment.

The next section describes an architecture addressing scenarios 1 and 2 and requirements 1 to 5, while scenario 3 is left for future work.

3 ARCHITECTURE

We provide support for scenarios 1 and 2 using the architecture shown in Fig. 4. A DSL is made of abstract syntax (the elements of interest, their properties and relations), concrete syntax (their visualization) and semantics (what the models mean, typically enacted by model simulators or code generators). In MDE, the abstract syntax of a DSL is described through a meta-model. Regarding the concrete syntax, in this work we focus on graphical DSLs, which are represented in a graph-like way.

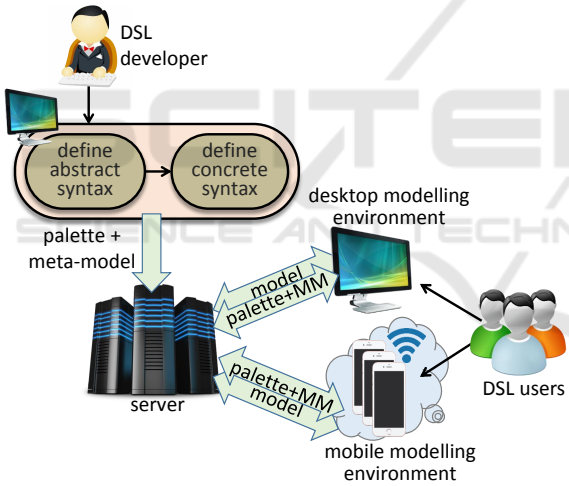


Figure 4: Proposed architecture.

To enable combined modelling, the same definition (abstract syntax meta-model and concrete syntax description) is used to configure both a desktop and a mobile modelling environment. Both the meta-model and the graphical syntax description are built in a desktop application (under Eclipse). We use the Eclipse Modelling Framework (EMF) (Steinberg et al., 2008) to create the meta-model, and our own tool to describe the graphical syntax (Garmendia et al., 2015). The concrete syntax description (which we call the *palette*) is defined using a wizard that allows the user to assign icons and shapes to the different meta-model elements. This palette is stored in the form of a model, too.

The DSL description can be uploaded to the server, and be used to synthesize both mobile and desktop modelling environments. In both cases, models can be stored locally or in the model server. This permits the seamless editing of models both in the mobile device and the desktop environment. Moreover, in case of mobility, it is also possible to set up a collaborative modelling session between several nearby users by temporarily designating one of their mobile devices as a local server. This enables collaboration without requiring an internet connection.

Once we have seen the main parts of the architecture we propose, in the next section we detail its main features. As a running example, we will use a DSL for the home networking domain.

4 TOOL SUPPORT

This section describes our prototype tool for the proposed architecture. The tool, named *DSL-comet* (Domain Specific Language COllaborative Modelling EnvironmenT) is made of three components: a desktop client, a server, and a mobile app. The desktop client is based on Eclipse, the server on Node.js, and the client is a native iOS app. We next explain the three components. More information about the tool is available at http://miso.es/tools/DSL-comet.html.

4.1 The Desktop Client

In order to define the abstract syntax of the DSL, we use our DSL-tao tool (Pescador et al., 2015). The distinguishing feature of the tool is that it permits the construction of the meta-model reusing predefined patterns. The back of Fig. 5 shows an excerpt of the meta-model of the home networking example.

Once the meta-model is ready, we need to establish the concrete syntax for the DSL. This is performed by a dedicated wizard, shown at the front of Fig. 5. The concrete syntax specification is described through a model that annotates the meta-model. Both the meta-model and the concrete syntax description can be uploaded to the server, from where they can be downloaded by mobile and desktop clients to produce a modelling environment.

We currently target Sirius for the desktop environment, but other technologies could be targeted as well. Fig. 6 shows a screenshot of the resulting editor. The view “Mobile Server View” permits downloading models and DSL modelling environments from the server, like the ones shown in the figure.

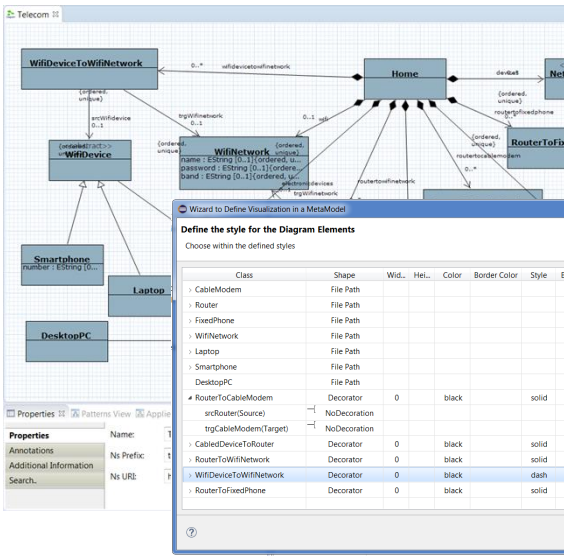


Figure 5: Meta-model for the home networking domain (back). Wizard to define graphical representation (front).

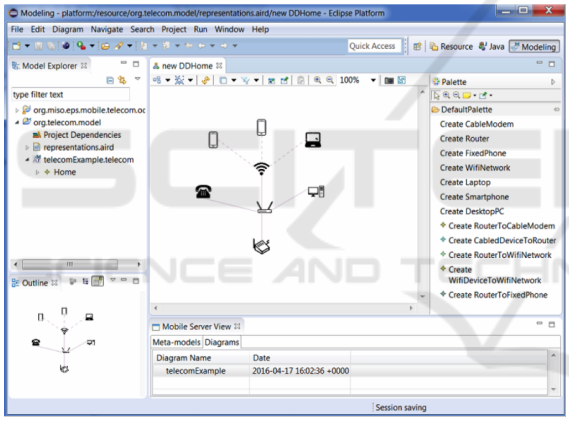


Figure 6: Screenshot of the Sirius desktop client.

4.2 The Server

We have developed a remote server in order to store models, meta-models and palettes. It is implemented using "Node.js" (Node.js, 2016), and it is deployed on the Heroku (Heroku, 2016) platform. The server can be accessed from http://miso.es/tools/DSL-comet.html.

There are two ways to interact with the server. Artefacts (models, meta-models, palettes) can be uploaded and downloaded using either REST services or a web application. To enhance scalability, artefacts are stored in a Mongodb (MongoDB, 2016) database using JSON format. As the desktop clients use EMF technology, we have developed a service that converts from the EMF format (i.e., XMI) to JSON, and vice versa. This technique has the advantage of providing a lighter, portable format for models to be used in the mobile apps. This exporting tool can be accessed as a server service and may be invoked from other applications.

4.3 The iOS Client

The mobile client has been developed for iOS devices (i.e., iPhones and iPads). It has been designed to use the minimum internet traffic, so that most of the time, it can be used with no data connectivity. An internet connection is necessary in order to download palettes and meta-models from the server. Once those files are downloaded, the DSL user may edit diagrams with no need of internet connection.

Fig. 7 shows the main screen of the mobile app, where the same model shown in Fig. 6 is being edited. The image is decorated with labels depicting the main functionalities.

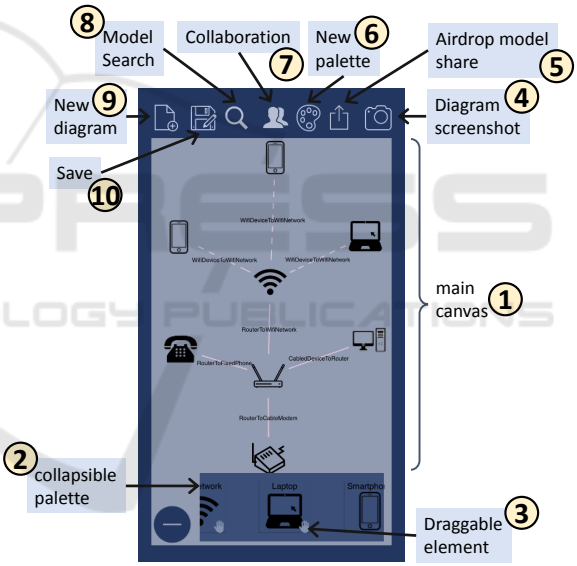


Figure 7: Screenshot of the editor on an iPhone 5S.

Label 1 corresponds to the canvas where the model is drawn. We have specially kept in mind the reduced screen size of mobile devices when creating the app (from 4.0 inches in an iPhone 5s, to 12.9 inches in an iPad Air). The user may drag classes from the bottom palette (label 2) to the canvas, in order to create instances of the classes. To save space, the palette can be hidden. Furthermore, we distinguish between draggable elements (label 3) which are identified by a small hand, from non-draggable ones which are identified by a finger. The latter ones cannot be dragged to the canvas, but they are created using a separate view in order to avoid cluttering the canvas.

Users may take a screenshot of the current model

(label 4 in the figure), which can be saved to the camera roll, or sent via Twitter or e-mail. Additionally, models can be shared via Airdrop[1] (label 5) or uploaded to Google Drive.

At any point, the user can select a new palette (label 6), search a model element (label 8), start a new blank model (label 9), or save a model locally or in the server (label 10). Palettes can be also stored in the server or locally. Finally, users may initiate a collaboration with nearby users (label 7). Next, we detail the model editing and collaboration facilities.

4.3.1 Mobile Model Editing

Model editing is done by gestures in the mobile touch screen. First, draggable elements are created by drag&drop from the palette. As the palette may be too long, we support scroll to show more elements. The canvas itself supports focus (double tap), zoom-in (pinch open) and zoom-out (pinch close).

Connecting elements is done by a long press from the source node to the target one. The tool is able to resolve the admissible relationships that may exist between those two elements. If several relationships are possible, the user can select the desired one.

If an instance is selected in the canvas, the application displays a detailed view with its attributes and output connections (see Fig. 8(a)). The user can update its attributes and the visual representation gets updated accordingly. Given that models can become large, the application includes a search tool where the user can ask for any entity using filters, as shown in Fig. 8(b). The filters allow searching for nodes having a certain value in some selected attributes.

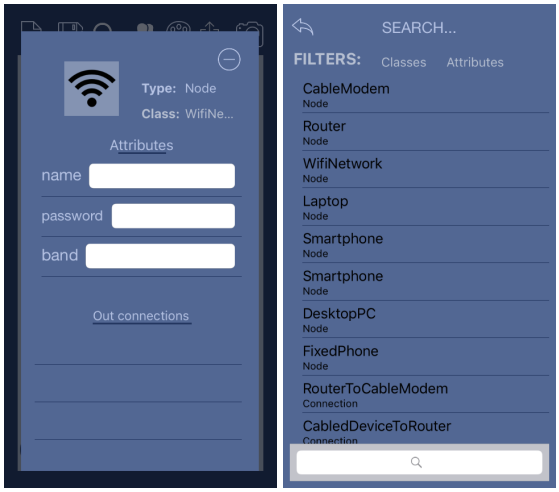
When the DSL developer defines the graphic representation for a meta-model, it is possible to set some classes as "Non-draggable". Those classes cannot be dragged from the palette. Instead, the only way to instantiate them is to select the class from the palette and manually create an instance. Later, the created instances can be changed and referenced from other (draggable or non-draggable) objects. Non-draggable elements enable more concise diagrams, as the DSL user can manipulate hidden instances that do not take diagram space. Fig. 9(a) shows how to create a non-draggable entity using a table. Fig. 9(b) depicts how the created element can be used, e.g., as the target of a relationship.

4.3.2 Collaboration Support

The mobile app allows a group of nearby users to work together on a diagram without an internet connection. For this purpose, first, one of the users needs to offer a diagram in collaboration. The user's device will become the *local server* of the session. Then, one or more users can connect to this local server. The role of this server is to store the diagram information and send the model changes to the clients periodically, so that every connected device has a synchronized model status.

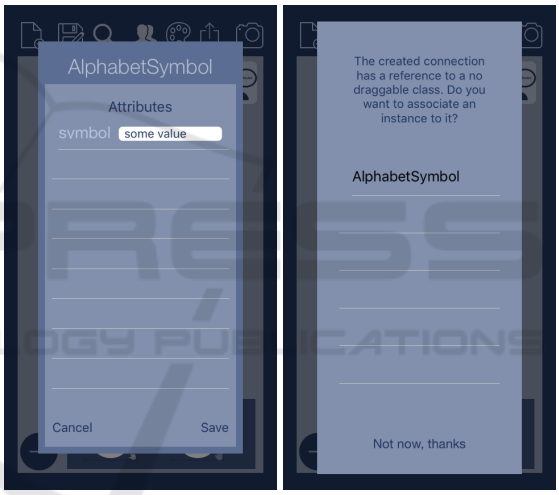
We use a token-based collaboration approach, where only the user holding the token (initially the server) can modify the model. The model changes are sent to the server device, and from there, they are propagated to all connected clients. Clients may ask for the token at any point, and the collaboration server has to agree (or deny) to grant the token.

[1] Airdrop is a file sharing technology of iOS similar to Bluetooth.



(a) (b)

Figure 8: (a) Node details. (b) Search facility.



(a) (b)

Figure 9: (a) Creating an instance of a class. (b) Associating that instance to a connection.

5 RELATED WORK

Although MDE has been used to produce mobile applications (Vaupel et al., 2014), few works report on mobile domain-specific modelling environments.

CEL is a mobile iOS application to create UML class diagrams (Lemma et al., 2013), with no support for collaboration or model sharing. FlexiSketch is a sketching mobile modelling tool especially tailored for software requirements modelling (Wüest et al., 2013), and it supports collaboration. However, none of these two tools support combined modelling in desktop and mobile.

Some works advocate the use of the web and the web browser for modelling (Rose et al., 2012). While these systems can be used in a mobile device, a dedicated mobile modelling environment permits a better optimization of the reduced screen size, interaction through typical mobile interaction gestures, and the possibility to work without internet connectivity.

The flexibility that touch screens provide for modelling has also been explored. Calico is a flexible modelling tool based on sketches. It works on a digital whiteboard, not on mobiles, but relies on touch-based interaction (Mangano et al., 2014).

Some works allow programming in the mobile using graphical languages (Danado and Paternò, 2014). However, such languages are fixed, and the environment is created ad-hoc for them. Instead, we enable the creation of arbitrary graphical DSLs, where their environment is configured with the DSL descriptions.

Altogether, our approach is novel as it permits creating both a desktop and a mobile DSL modelling environment, combined modelling in the mobile and the desktop, and collaboration using mobile devices.

6 CONCLUSIONS

This paper has presented our proposal for enabling mobile domain-specific modelling, showing some scenarios of interest and a working prototype tool. We claim that enabling modelling on mobile devices present interesting opportunities for MDE, including more flexibility and the use of contextual information.

We are currently improving our prototype tool to support more advanced collaborative model editing. In the short term, we will also address scenario 3 related to contextual modelling. Finally, we plan to conduct user studies to evaluate our proposal for different applications.

ACKNOWLEDGEMENTS

Work supported by the Spanish Ministry of Economy and Competitivity (TIN2014-52129-R), the R&D programme of the Madrid Region (S2013/ICE-3006), and the EU commission (FP7-ICT-2013-10, #611125).

REFERENCES

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., and Riboni, D. (2010). A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.*, 6(2):161–180.

Brambilla, M., Cabot, J., and Wimmer, M. (2012). *Model-Driven Software Engineering in Practice*. Morgan & Claypool, USA.

Danado, J. and Paternò, F. (2014). Puzzle: A mobile application development environment using a jigsaw metaphor. *J. Vis. Lang. Comput.*, 25(4):297–315.

Garmendia, A., Pescador, A., Guerra, E., and de Lara, J. (2015). Towards the generation of graphical modelling environments aided by patterns. In *SLATE*, CCIS, pages 1–8. Springer.

Heroku (2016). https://www.heroku.com/.

Kelly, S. and Tolvanen, J. (2008). *Domain-Specific Modeling - Enabling Full Code Generation*. Wiley.

Lemma, R., Lanza, M., and Olivero, F. (2013). CEL: modeling everywhere. In *ICSE*, pages 1323–1326. IEEE / ACM.

Mangano, N., LaToza, T. D., Petre, M., and van der Hoek, A. (2014). Supporting informal design with interactive whiteboards. In *CHI*, pages 331–340. ACM.

MongoDB (2016). https://www.mongodb.org/.

Node.js (2016). https://nodejs.org/.

Pescador, A., Garmendia, A., Guerra, E., Cuadrado, J. S., and de Lara, J. (2015). Pattern-based development of domain-specific modelling languages. In *MoDELS*, pages 166–175. IEEE.

Rose, L. M., Kolovos, D. S., and Paige, R. F. (2012). Eugenia live: A flexible graphical modelling tool. In *XM*, pages 15–20. ACM.

Sirius (2016). https://eclipse.org/sirius/.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2008). *EMF: Eclipse Modeling Framework, 2nd Edition*. Addison-Wesley Professional, NJ.

Vaupel, S., Taentzer, G., Harries, J. P., Stroh, R., Gerlach, R., and Guckert, M. (2014). Model-driven development of mobile applications allowing role-driven variants. In *MODELS*, volume 8767 of *LNCS*, pages 1–17. Springer.

Wüest, D., Seyff, N., and Glinz, M. (2013). Flexisketch: A mobile sketching tool for software modeling. In *MobiCASE*, volume 110 of *LNICST*, pages 225–244. Springer.