Domain-Specific Modelling Using Mobile Devices

Diego Vaquero-Melchor, Antonio Garmendia, Esther Guerra(✉), and Juan de Lara

Modelling and Software Engineering Group, Computer Science Department, Universidad Autónoma de Madrid, Madrid, Spain
{diego.vaquero,antonio.garmendia,esther.guerra,juan.delara}@uam.es
http://miso.es

Abstract. Domain-Specific Languages (DSLs) are languages tailored for a specific application area, like logistics, networking or mobile app design. They capture the main primitives and abstractions within a domain, which permits modelling systems and problems within that domain in a succinct and natural way. DSLs are heavily used in software development paradigms like Model-Driven Engineering, and they are also a means to enable end-users to perform simple programming tasks in particular domains.

Traditionally, modelling using DSLs has been supported by desktop computers in static settings that neglect the surrounding contextual information. Instead, we claim that DSLs can also be very useful in a dynamic setting where they can profit from mobility and context. Therefore, in this paper, we identify several scenarios where modelling using mobile devices – like smartphones or tablets – is useful. We also propose an architecture and a tool, called *DSL-comet*, which enables mobile modelling using graphical DSLs, and supports seamless integration of desktop and mobile graphical modelling environments.

Keywords: Model-Driven Engineering · Domain-Specific Languages · Graphical modelling languages · Context · Mobile devices · DSL-comet

1 Introduction

Domain-Specific Languages (DSLs) [1,2] are "small" languages tailored to a particular domain. In contrast to general-purpose languages (GPLs) – like Java for programming or UML for modelling – DSLs target specific application areas, like networking, user interface design or logical circuits [1]. This way, DSLs provide useful primitives of the domain, which can be used to create simpler, more intentional system descriptions than those that would result from the use of GPLs. DSLs can be either graphical [1] or textual [3], though in this paper we will focus on graphical ones.

DSLs are heavily used in Model-Driven Engineering (MDE) [4], a software engineering paradigm that promotes an active utilization of models in all phases

E. Cabello et al. (Eds.): ICSOFT 2016, CCIS 743, pp. 221–238, 2017.
DOI: 10.1007/978-3-319-62569-0_11

of software development. In MDE, models are used to specify, analyse, simulate, test, execute and generate code for the final applications, among other activities. While it is possible to define these models using GPLs like the UML, their construction using DSLs tailored to particular domains is very frequent in practice [1,5]. DSLs are also enablers for end-user development [6], as they permit users with no or little computer science background to perform concrete, simple programming tasks in particular contexts.

A primary goal of models in MDE is to serve as an automation mechanism for development tasks like code generation. Thus, although 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, the modelling task takes place in desktop computers (or laptops) assisted by modelling tools, like those based in Eclipse/EMF [7]. While this is useful for late phases of model development, it introduces rigidity and prevents using models in flexible scenarios that imply mobility and collaboration or need to react to contextual information. Unfortunately, most tools for the creation of DSLs are targeted to desktop environments [1,3].

We claim that modelling using DSLs can benefit from mobility, collaboration and context in several situations. In this paper, we identify scenarios where mobile modelling is useful, and present an architecture and prototype tool for the discussed scenarios. 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. Our desktop modelling environment is an Eclipse plugin based on the Sirius [8] graphical modelling platform. The mobile modelling environment is based on iOS, and permits model sharing and local collaborative model editing via local ad-hoc WiFi networks. Communication between the desktop and mobile environments is achieved through a dedicated server. Our tool is called *DSL-comet* (Domain Specific Language COllaborative Modelling EnvironmenT) and is freely available at the Apple's app store, and at http://miso.es/tools/DSL-comet.html. To illustrate its functionality, we will introduce a DSL for designing factory plants as a running example.

This is an extended version of our previous paper [9] presented at the 10^{th} International Joint Conference on Software Technologies (ICSOFT). In this paper, we enhance the presentation of the motivating scenarios for mobile modelling, we present a more comprehensive description of the technical aspects of our tool and architecture using a different case study, and we expand the analysis of related works.

The rest of this paper is organized as follows. First, Sect. 2 motivates the need for mobile modelling using DSLs, describes several scenarios of interest, and elicits some technical requirements for tools aimed at supporting mobile modelling. Next, Sect. 3 describes the architecture we propose to support these scenarios. Section 4 introduces *DSL-comet*, the prototype tool that realizes this vision. Then, Sect. 5 presents a comparison with related research. Finally, Sect. 6 ends with the conclusions and open lines of future work.

2 Scenarios for Mobile Modelling

In this section, we discuss several scenarios where modelling can profit from mobility, context and collaboration. We will use these scenarios to elicit requirements for tools aimed at supporting domain-specific modelling in mobile devices.

2.1 Multi-device Modelling

Mobile modelling tools should keep models compatible with other devices. Therefore, in the first scenario, we deem necessary being able to use seamlessly models both in mobile and desktop environments. This means that models can be created in a desktop environment and then be used in a mobile device, or vice-versa. Figure 1 shows a schema of this scenario. A server is in charge of storing the models, which can be downloaded for their editing in mobile and desktop environments indistinctly, and then be uploaded to the server again.

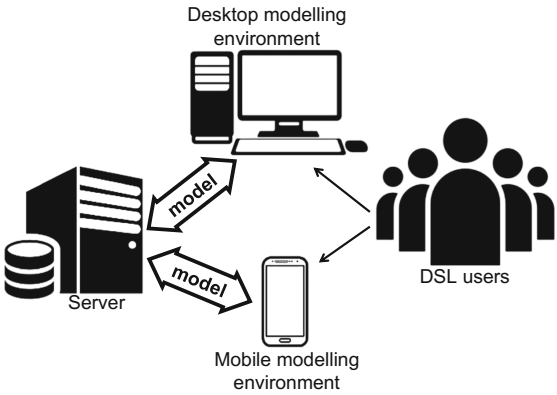

Fig. 1. Combined desktop and mobile modelling.

Applications of this scenario include modelling in remote locations (e.g., a wind turbine) through mobile devices. As an example, an operator of a factory may need to inspect a model of the factory plant on-site, change its parameters according to the current working location, or even create a model of the plant while visiting the factory. In this case, the operator would find preferable performing these modelling actions using a mobile device while staying on the plant. On the other hand, the same models may need to be analysed by other engineers at desktops in the company offices, or be used for simulation or run-time monitoring.

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 is also applicable to the educational domain. In this setting, professors can create models and modelling exercises in their desktop computers, and students may access these exercises or modelling lessons for learning in mobility. Then, students may upload the solution to the exercises to the server, and be graded by the professors in their desktops.

From the analysis of this scenario, we derive as a technical requirement the need of a common format to represent models in desktop and mobile environments. This is more easily achievable if both environments are generated from a single definition of the DSL being used to build the models.

2.2 Mobile Collaborative Modelling

When modelling in mobility in remote locations (e.g., a farm or a building in the country side) one cannot assume the availability of a WiFi Internet connection or even mobile coverage. In this scenario, users can benefit from the short-range communication capabilities of mobile devices to enable *local* collaboration, e.g., for joint model construction or inspection. This eliminates the need to use a remote server to orchestrate and coordinate the collaboration, which may incur in long delays or can be impossible in remote locations where no data connection is available. Instead, collaboration can occur by using short-range communication of mobile devices like Bluetooth or WiFi.

Figure 2 illustrates this scenario. First, one user (user 1) downloads from the server a palette with the different kinds of elements that can appear in the model. Alternatively, the user may already have the palette stored locally in the mobile device. 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. For example, it can be token-based, with either implicit or explicit assignment of the modification token. The figure shows a token-based collaboration, where only the user holding the modification

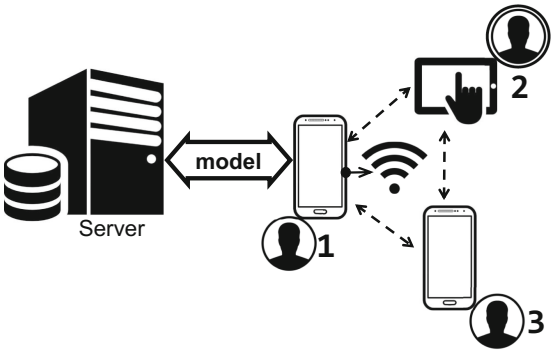

Fig. 2. Local collaborative modelling. User 1 starts the session, and user 2 has the modification token. All users' devices display an updated view of the model as it is being modified.

token (user 2) can change the model. In the meantime, the devices of all users participating in the session will display an updated view of the model as it is being modified. When the session finishes, the model can be stored either in the server or locally.

Applications of this scenario include those presented in Sect. 2.1, but enhanced with local collaboration facilities. In particular, local collaboration enables joint model creation, model revision and discussion, or the collaborative solution of modelling exercises in an educational setting.

2.3 Context-Based Modelling

Mobile devices can access contextual information, which can be useful in some modelling scenarios [10]. For example, a mobile modelling environment may present different parts of a model, or allow different editing actions, depending on the context. This context may include information about the device state – like battery, size of screen, orientation, or availability of a WiFi connection – and external information – like position, time or weather conditions –. Figure 3 illustrates the adaptation of a mobile modelling environment depending on the context.

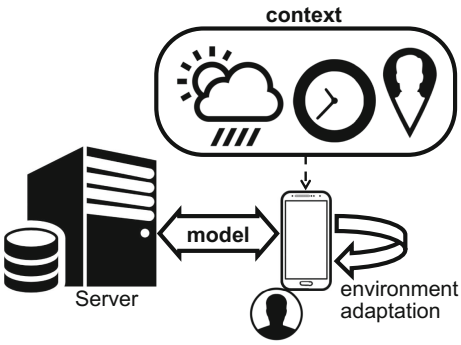

Fig. 3. Mobile contextual modelling.

For instance, in the factory example introduced in Sect. 2.1, the mobile app may present a model of the plant where the engineer is located, updating this view when the engineer moves to a different location. This way, it becomes easier for the engineer to monitor or modify the operating conditions of the machines nearby the current location.

Domotics is another domain where this scenario applies. Similarly to the factory example, a mobile app could present a model of the devices (TVs, blinds, lights, heating, etc.) inside the room where the house owner is currently located. This view would get updated when the owner moves to a different room. By manipulating the model, the owner may interact with the home devices.

2.4 Requirements for Mobile Domain-Specific Modelling

From an analysis of the presented scenarios, we identify the following requirements for a mobile modelling platform:

Rq1. Models should be compatible in mobile and desktop applications. The environment should enable the seamless use of models in desktop and mobile devices.
Rq2. In order to allow multi-device modelling (scenario 1), the generation of both desktop and mobile environments should be easy. Moreover, the effort needed to generate one desktop and one mobile environment should be the same as the effort needed to generate only one of them.
Rq3. Model visualization in the mobile environment should be adapted to the reduced screen size.
Rq4. Model editing in the mobile environment should be adapted to support typical mobile interaction gestures (e.g., swipe, tap and pinch).
Rq5. In order to allow mobile collaborative modelling (scenario 2), the platform should support local collaboration in the mobile environment.
Rq6. In order to allow context-based modelling (scenario 3), the platform should enable context adaptation in the mobile environment, and incorporation of context information and the corresponding adaptation rules in the DSL definition.

The next section describes an architecture that addresses scenarios 1 and 2 and requirements Rq1 to Rq5. Scenario 3 and requirement Rq6 are left for future work.

3 Architecture

Figure 4 shows the scheme of our proposed architecture, which provides support for scenarios 1 and 2. It considers two main phases: DSL definition (label 1) and DSL use (label 2).

In the first phase, the DSL developer defines the DSL. This includes the definition of the DSL abstract syntax (the concepts of interest, together with their properties and relations), concrete syntax (their visualization), and semantics (what the models mean, typically enacted by model simulators or code generators). In this work, we focus on the abstract and concrete syntax, and leave the semantics for future work.

In MDE, the abstract syntax of a DSL is described through a meta-model. Implementation-wise, we use standard tools based on the Eclipse Modelling Framework (EMF) [7] to create the meta-models. Therefore, meta-models are built using Eclipse in a desktop environment, and then they are uploaded to the server once their definition is complete.

The graphical concrete syntax (which we call palette) can be defined either from the mobile environment or from Eclipse in the desktop [11]. In both cases, it is defined using a wizard that allows assigning icons and shapes to the different

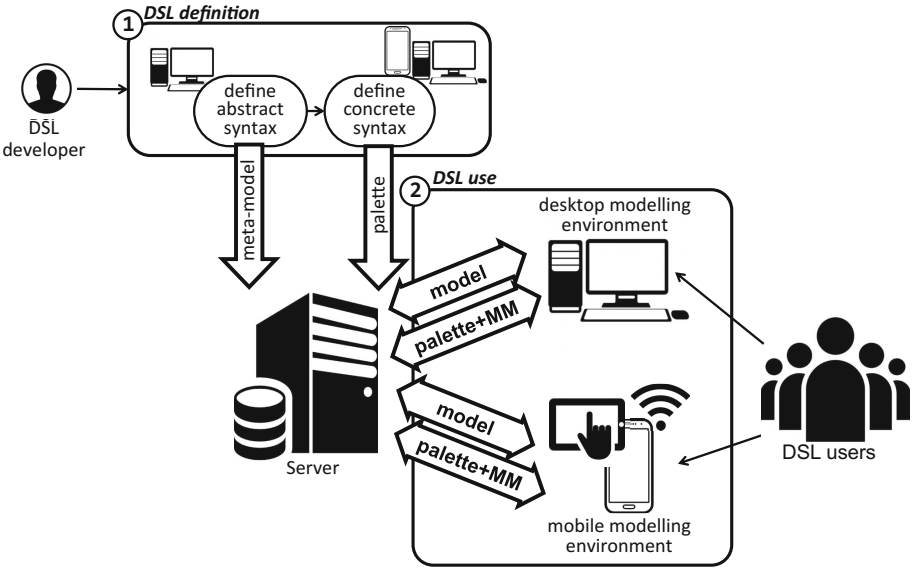

Fig. 4. Our proposed architecture.

meta-model elements. This palette is stored as a model in the server. For this purpose, we have created a meta-model to describe graphical concrete syntaxes in a platform-independent way.

To enable combined modelling, we use the same definition (abstract syntax meta-model and concrete syntax description) to synthesize both a desktop and a mobile modelling environment, thus covering requirements Rq1 and Rq2. The desktop environment is realised as a Sirius editor, while we have built our own tool called *DSL-comet* to allow the editing of models in mobile devices. *DSL-comet* supports typical visualization and interaction styles for mobile devices, thus covering requirements Rq3 and Rq4.

The DSL users can build models using any of the two generated environments, and store the models either locally or in the server. This permits the seamless editing of models both in the mobile device and the desktop environment. 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, as demanded by requirement Rq5.

Currently, we use a MongoDB NoSQL database to store the models, meta-models and palettes. Technically, all these artefacts are stored in JSON format, and they are converted into XMI to ensure compatibility with the desktop environment.

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 factory plants.

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 is based on Node.js, and the client is a native iOS app. Next, we 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 tool DSL-tao [12]. The distinguishing feature of the tool is that it permits constructing meta-models by composing predefined patterns. As an example, the window at the back of Fig. 5 shows an excerpt of the meta-model for the factory DSL, specified using DSL-tao. According to the meta-model, a factory may contain different types of machines (generators, terminators and assemblers) connected through conveyors and controlled by operators. Machines manipulate different types of parts, like handles, knobs and hammers. Moreover, two attributes in class Machine permit configuring whether a machine is busy or broken.

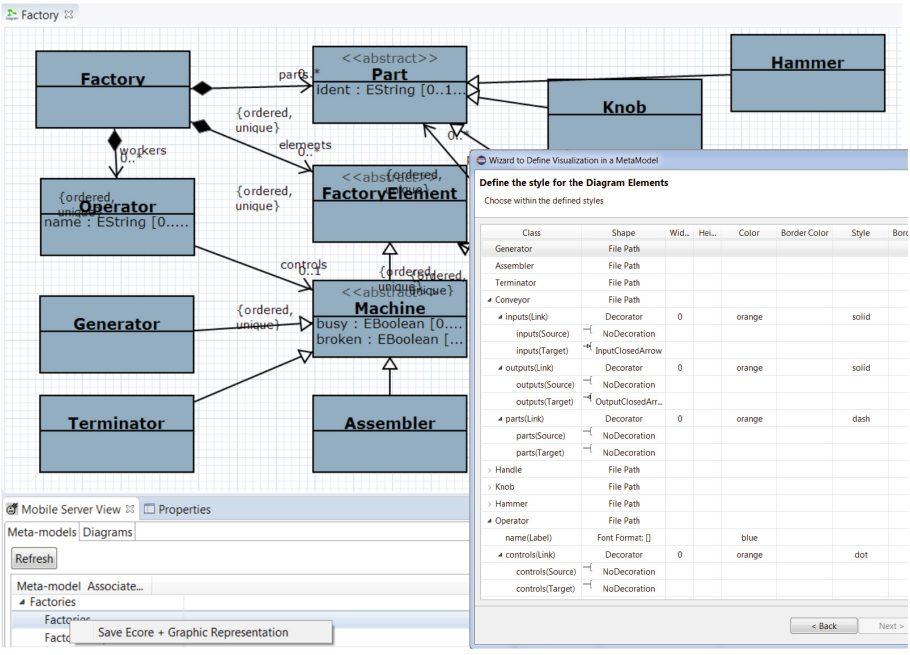

Fig. 5. Meta-model of a DSL for factory plants (back). Wizard to define the concrete syntax of the DSL (front).

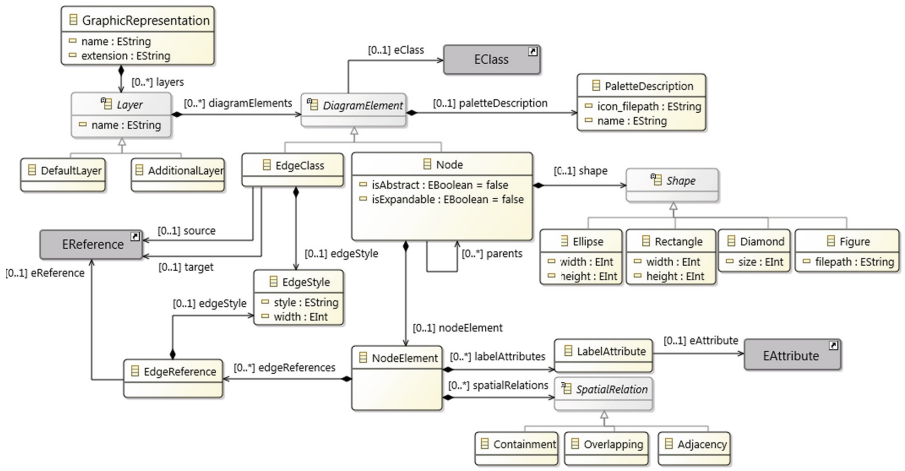

Fig. 6. Excerpt of *GraphicRepresentation* meta-model.

Once the meta-model of the DSL is complete, we need to define its concrete syntax. This is performed using a dedicated wizard, which is shown at the front of Fig. 5. The specified concrete syntax is internally described through a platform-independent meta-model called *GraphicRepresentation* that records the selected appearance for the classes and relations in the DSL meta-model (shape, colour, etc.) as well as the palette that the generated modelling environments will provide to create instances of the different classes. An excerpt of the meta-model is shown in Fig. 6. The diagram elements are organized into layers. This is useful to occlude elements or show more details. However while the generated desktop graphical editor supports layers, currently the mobile editor does not.

Layers contain graphical elements (class DiagramElement), which point to the meta-model classes they represent. Objects can be represented as nodes (class Node) or edges (class EdgeClass). Class EdgeClass provides attributes for setting the references acting as the source or target of the edge, from the available references of the class. Moreover, the EdgeStyle contain the information about the graphical style of the edge (e.g., dash, dot or solid). With respect to the Nodes, there are different styles of representation, either defining a predefined figure (e.g., Ellipse, Rectangle or Diamond) or an external one (e.g., SVG). Some of its attributes can be selected as the label of the node (class LabelAttribute), and spatial relations between nodes (adjacency, overlapping, containment) can be defined. However currently, the mobile editor does not support spatial relations, but the desktop editor does. Common graphical descriptions among different objects can be reused using abstract nodes (attribute isAbstract in class Node) and inheritance relations (references Node.parents). As we will see in Sect. 4.3, "expandable" nodes (Nodes with isExpandable true) can include non-graphical objects, especially useful in mobile devices because of their reduced screen size (Rq3 in Sect. 2.4).

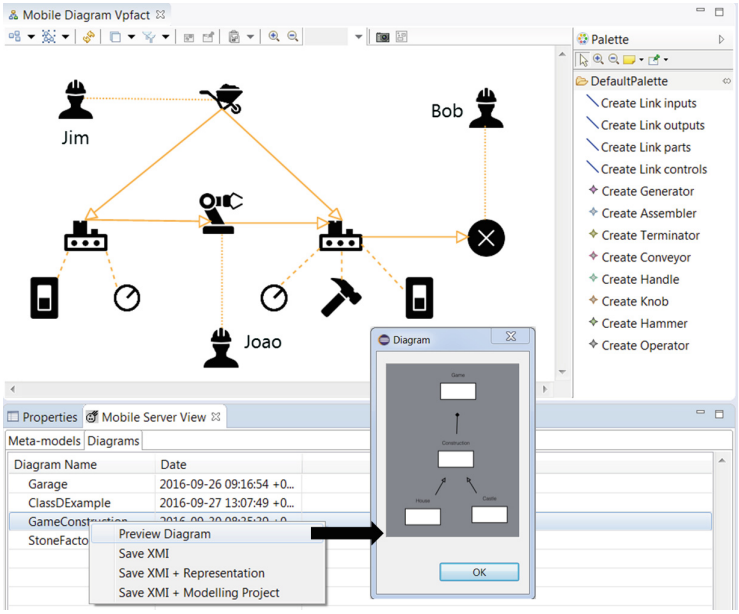

Fig. 7. Screenshot of the Sirius desktop client.

From the DSL meta-model and the concrete syntax description, we generate a desktop graphical modelling environment based on Sirius. Although we currently target Sirius, other technologies like Graphiti [13] or EuGENia [14] could be targeted as well. Figure 7 shows a screenshot of the resulting desktop editor. Since it is a Sirius-based editor, its generation accounts to producing an *odesign* model that describes the modelling workbench.

The desktop client provides a dedicated view called *Mobile Server View* to interact with the remote server. This view has two tabs. The first one, named *Meta-models*, lists all meta-models stored in the server and its corresponding graphic representation. The window in the back of Fig. 5 shows this tab. By right clicking on one of the listed meta-models it is possible to download it, together with its concrete syntax. Additionally, the client has also functionalities to upload the meta-model and the concrete syntax in the server. We allow a single meta-model to have several concrete syntax representations.

Figure 7 shows the second tab of the view, called *Diagrams*. It displays the list of diagrams (i.e., models) stored in the server. By right clicking on a model, a contextual menu with several options appears. From the options in this menu, it is possible to obtain a pre-visualization of the selected model (as shown in the figure), download the model in XMI format (standard format to persist EMF models), and download the graphical information of the model (e.g., position of nodes). As an example, Fig. 7 shows a model previously created in a mobile device, which has been downloaded (both its XMI and graphical information) to the desktop environment from the server. Sirius stores the graphical information attached to models in *aird* files.

4.2 The Server

We have developed a remote server to store models, meta-models and palettes (called artefacts henceforth). The server is deployed on the Heroku [15] platform and uses "Node.js" [16] technology. The server can be accessed from http://miso.es/tools/DSL-comet.html.

There are two ways to manage artefacts in the server: either using REST services or through the web-based application. In order to enhance scalability, we store artefacts in a MongoDB [17] database using JSON format. The advantage this format brings is that other external tools can use our artefacts directly.

On the other hand, the desktop clients use EMF technology, which is not directly accessible on mobile platforms. To solve this problem, we have developed some services to convert back and forth between JSON and XMI. This technique has the advantage of providing a lighter, portable format for using models in mobile apps. Other mobile platforms may use these services to work with meta-models.

4.3 The iOS Client

The mobile client has been developed for iOS devices (i.e., iPhone and iPad). It has been designed to use the minimum internet traffic, and therefore, it does not require data connectivity most of the time. An internet connection is only necessary to download palettes and meta-models from the server though. Once those files are downloaded, the user may create and edit diagrams with no need for connectivity.

Figure 8 shows the main screen of the mobile app, where the same model shown in Fig. 7 is being edited. The image is decorated with labels depicting its main functionalities.

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 in. in an iPhone 5s, to 12.9 in. in an iPad Air). The user may drag classes from the bottom palette (label 2) to the canvas, in order to create instances of them. This palette can be collapsed to save space.

The user can add annotations (such as notes, hand-made drawings and temporal alerts) to the diagram, using the button with label 4. A new model can be created (label 5) and saved locally or in the server (label 6).

Label 7 points to the search tool, which is useful to find elements on the canvas using filters. Users may initiate a collaboration session with nearby users (label 8), select a new palette (label 9) and share the model via Airdrop[1]. Finally, it is possible to take a screenshot of the current model (label 11) and save it on the camera roll or send it via Twitter or e-mail.

Tool Workflow. The app user can either use a palette from the server or use a local one (see Fig. 9). Taking into account that this tool may be used without internet connection, the app can download a palette and store it locally on the mobile device.

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

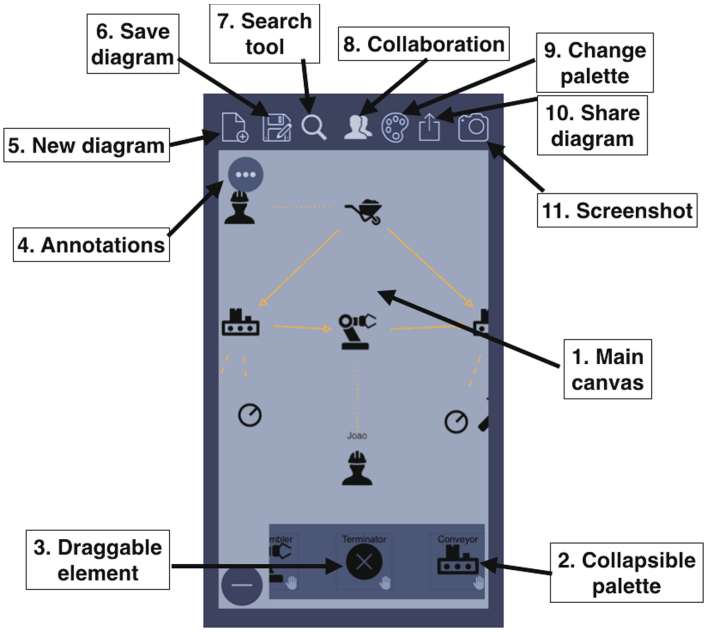

Fig. 8. Screenshot of the editor on an iPhone SE.

Model editing is done by gestures in the mobile touch screen. Draggable elements are created by dragging from the palette. As the palette may be too long, we support scrolling to show more elements. The canvas itself supports zoom-in (open pinch) and zoom-out (close pinch).

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 object is selected in the canvas, the application displays a detailed view with its attributes and output connections (see Fig. 10). 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 object using filters, as shown in Fig. 11. The filters allow searching for nodes having a certain value in some selected attributes, as shown in Fig. 12.

When defining a DSL (see Sect. 4.1), the DSL developer can declare some references as "Expandable". Figure 13 shows an instance of the Conveyor class with the default behavior, which is representing the links between the conveyor and the three parts it contains (a hammer, a handle and a knob) as edges. By setting the part reference of the Conveyor class as "Expandable", those parts would not be shown in the canvas. Instead, the details view of the conveyor would include an option to create instances of Part, as Fig. 14 shows. If we select the *Create Link Parts* option, the new view in Fig. 15 will be shown. From this view, we can create instances associated to the conveyor that will not appear

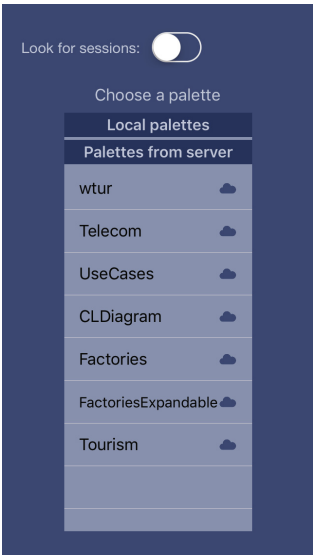

Fig. 9. Selecting a palette.

Fig. 10. Example of a node details.

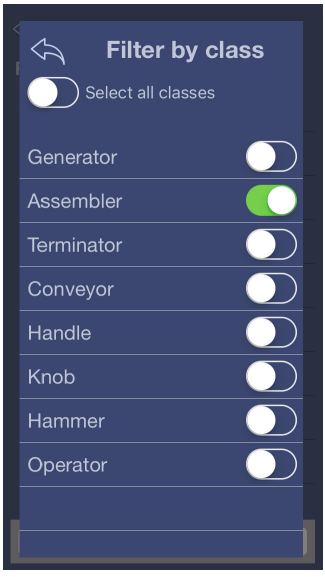

Fig. 11. Filtering by class type.

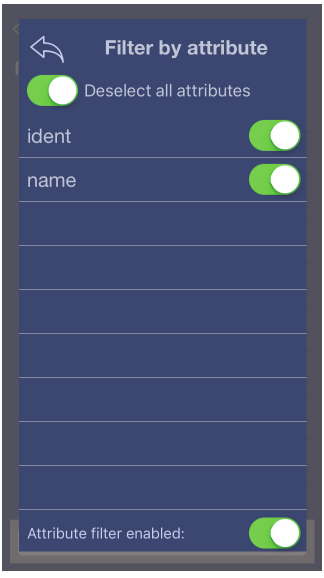

Fig. 12. Filtering by attributes.

on the canvas. This is especially useful in a mobile app to save space, given the reduced size of the screen of mobile devices.

Once the user has created the model, it can be saved either on the server or locally. The model can also be shared via Airdrop or via some external apps

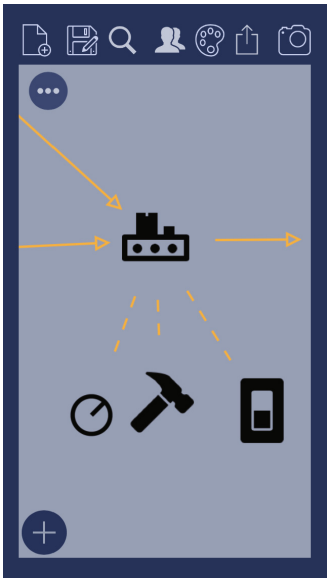

Fig. 13. Example of a conveyor without expandable items.

Fig. 14. A conveyor with an expandable reference.

Fig. 15. Creating a part linked to the conveyor.

like Google Drive or Dropbox (Fig. 16). The model is serialized as a ".demiso" file with XML schema. This file extension is detected by the operative system,

whereby it will show the user the option to open those files with the *DSL-comet* app.

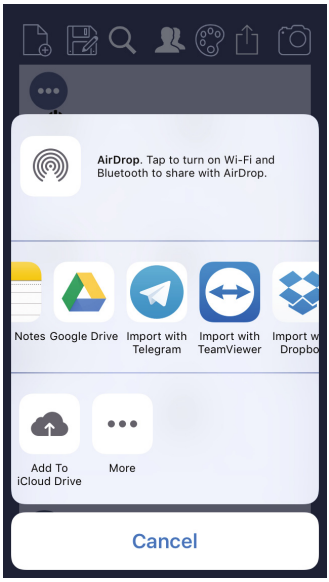

Fig. 16. Some sharing options.

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. Any model change is sent to the server device, and from there, it is propagated to all connected clients, so that they see the synchronized model on their screens. Clients may ask for the token at any point, and the collaboration server has to agree (or deny) to grant the token.

5 Related Work

Many tools have been proposed along the years to create graphical DSLs, like AToM^3 [18], EuGENia [14], GMF [19], Graphiti [13], MetaEdit+ [1] or Sirius [8]. However, most of them target the generation of graphical editors for the desktop, but not for mobile devices.

Some recent works allow creating graphical DSL environments for modelling in the web, like AToMPM [20], EuGENia Live [21] or WebGME [22]. However, although these environments can be utilized within a mobile device using a web browser, this poses several drawbacks. First, the web environments are not tailored to the particularities of mobile devices, whereas a mobile app is optimized for its execution in the mobile, and enables forms of visualization and interaction gestures especially designed for the reduced space of a mobile device. Second, web applications require connectivity, which might not be available when modelling in remote locations. Finally, relying on a web application for collaborative modelling might involve greater delays than the local short-range form of collaboration we support.

On the other hand, although MDE has been used to produce mobile applications [23], few works report on mobile domain-specific modelling environments. Some of them are described next.

CEL [24] is a mobile iOS application to create UML class diagrams, with no support for collaboration or model sharing. FlexiSketch [25] is a sketching mobile modelling tool especially tailored for software requirements modelling, and it supports collaboration. However, none of these two tools support combined modelling in desktop and mobile.

The flexibility that touch screens provide for modelling has also been explored. For instance, Calico [26] is a sketching tool, where the sketched elements can be scrapped and reused in other parts of the diagrams. It works on a digital whiteboard, not on mobiles, but relies on touch-based interaction.

Some works allow programming in mobile devices using graphical languages [27]. 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. We believe that our tool could greatly simplify the construction of these kinds of applications.

Altogether, we can conclude that our approach is novel as it permits creating both a desktop and a mobile DSL modelling environment, multi-device modelling in the mobile and the desktop, and collaboration using mobile devices.

6 Conclusions

In this paper, we have presented our proposal for enabling mobile domain-specific modelling, showing some scenarios of interest and a working prototype tool called *DSL-comet*. 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 and requirement Rq6 related to contextual modelling, which implies specifying the contextual information of interest and adaptation rules in the DSL definition. We would like to combine the tool with Wodel [28], a system to generate modelling

exercises, so that students can make those exercises in mobile devices. Finally, we plan to conduct empirical user studies to evaluate our proposal for different domains.

Acknowledgements. This work was supported by the Spanish Ministry of Economy and Competitivity (TIN2014-52129-R), and the R&D programme of the Madrid Region (S2013/ICE-3006).

References

1. Kelly, S., Tolvanen, J.: Domain-Specific Modeling - Enabling Full Code Generation. Wiley, Hoboken (2008)
2. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**, 316–344 (2005)
3. Voelter, M.: DSL Engineering - Designing, Implementing and Using Domain-Specific Languages (2013). dslbook.org
4. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Morgan & Claypool, San Rafael (2012)
5. Whittle, J., Hutchinson, J.E., Rouncefield, M.: The state of practice in model-driven engineering. IEEE Softw. **31**, 79–85 (2014)
6. Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A.F., Burnett, M.M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B.A., Rosson, M.B., Rothermel, G., Shaw, M., Wiedenbeck, S.: The state of the art in end-user software engineering. ACM Comput. Surv. **43**, 21 (2011)
7. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework, 2nd edn. Addison-Wesley Professional, Boston (2008)
8. Sirius (2016). https://eclipse.org/sirius/
9. Vaquero-Melchor, D., Garmendia, A., Guerra, E., de Lara, J.: Towards enabling mobile domain-specific modelling. In: ICSOFT 2016, vol. 2, pp. 117–122. ICSOFT-PT, SciTePress (2016)
10. Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D.: A survey of context modelling and reasoning techniques. Pervasive Mob. Comput. **6**, 161–180 (2010)
11. Garmendia, A., Pescador, A., Guerra, E., de Lara, J.: Towards the generation of graphical modelling environments aided by patterns. In: Sierra-Rodríguez, J.-L., Leal, J.P., Simões, A. (eds.) SLATE 2015. CCIS, vol. 563, pp. 160–168. Springer, Cham (2015). doi:10.1007/978-3-319-27653-3_16
12. Pescador, A., Garmendia, A., Guerra, E., Cuadrado, J.S., de Lara, J.: Pattern-based development of domain-specific modelling languages. In: MODELS, pp. 166–175. IEEE (2015)
13. Graphiti. https://eclipse.org/graphiti/
14. Kolovos, D.S., Rose, L.M., Abid, S.B., Paige, R.F., Polack, F.A.C., Botterweck, G.: Taming EMF and GMF using model transformation. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010. LNCS, vol. 6394, pp. 211–225. Springer, Heidelberg (2010). doi:10.1007/978-3-642-16145-2_15
15. Heroku (2016). https://www.heroku.com/
16. Node.js (2016). https://nodejs.org/
17. MongoDB (2016). https://www.mongodb.org/

18. de Lara, J., Vangheluwe, H.: AToM^3: a tool for multi-formalism and meta-modelling. In: Kutsche, R.D., Weber, H. (eds.) FASE 2002. LNCS, vol. 2306, pp. 174–188. Springer, Heidelberg (2002). doi:10.1007/3-540-45923-5_12
19. GMF. http://www.eclipse.org/modeling/gmp/
20. Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Mierlo, S.V., Ergin, H.: AToMPM: a web-based modeling environment. In: Joint Proceedings of MODELS Invited Talks, Demonstration Session, Poster Session, and ACM SRC, Proceedings of CEUR Workshop, vol. 1115, pp. 21–25 (2013). CEUR-WS.org
21. Rose, L.M., Kolovos, D.S., Paige, R.F.: Eugenia live: a flexible graphical modelling tool. In: XM @ MoDELS, pp. 15–20. ACM (2012)
22. Maróti, M., Kecskés, T., Kereskényi, R., Broll, B., Völgyesi, P., Jurácz, L., Levendovszky, T., Lédeczi, Á.: Next generation (meta)modeling: web- and cloud-based collaborative tool infrastructure. In: MPM @ MoDELS, Proceedings of CEUR Workshop, vol. 1237, pp. 41–60 (2014). CEUR-WS.org
23. Vaupel, S., Taentzer, G., Harries, J.P., Stroh, R., Gerlach, R., Guckert, M.: Model-driven development of mobile applications allowing role-driven variants. In: Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfran, E. (eds.) MODELS 2014. LNCS, vol. 8767, pp. 1–17. Springer, Cham (2014). doi:10.1007/978-3-319-11653-2_1
24. Lemma, R., Lanza, M., Olivero, F.: CEL: modeling everywhere. In: ICSE, pp. 1323–1326. IEEE/ACM (2013)
25. Wüest, D.: FlexiSketch: a mobile sketching tool for software modeling. In: Uhler, D., Mehta, K., Wong, J.L. (eds.) MobiCASE 2012. LNICST, vol. 110, pp. 225–244. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36632-1_13
26. Mangano, N., LaToza, T.D., Petre, M., van der Hoek, A.: Supporting informal design with interactive whiteboards. In: CHI, pp. 331–340. ACM (2014)
27. Danado, J., Paternò, F.: Puzzle: a mobile application development environment using a jigsaw metaphor. J. Vis. Lang. Comput. **25**, 297–315 (2014)
28. Gómez-Abajo, P., Guerra, E., de Lara, J.: Wodel: a domain-specific language for model mutation. In: SAC, pp. 1968–1973 (2016)