Active Domain-Specific Languages: *making every mobile user a modeller*

Diego Vaquero-Melchor, Javier Palomares, Esther Guerra, Juan de Lara
Modelling & Software Engineering Research Group (http://miso.es)
Universidad Autónoma de Madrid, Spain

***Abstract*—Domain-specific languages (DSLs) are small languages tailored to a certain application area, like logistics, web application testing or smart city planning. Traditionally, the use of DSLs has been limited to a static setting in desktop or web editors. However, in this paper, we claim that DSLs can be central components of mobile collaborative applications. In our vision, graphical DSLs can be extended to make use of mobility and context, and integrate heterogeneous information gathered from open APIs. We call this new generation languages "*active DSLs*".**

We foresee a range of scenarios where active DSLs can be useful. On the one hand, they can be used more flexibly in remote locations by enabling local collaboration of several mobile devices using their short-range communication capabilities. On the other hand, they can be extended with contextual features like geolocation, allowing the integration of maps and geo-services within the DSL, or the DSL rendering customization in response to contextual information. Active DSLs can also retrieve information from open APIs, in which case, models defined with the DSL become aggregators of heterogeneous data.

In this paper, we explain our vision for active DSLs and the first steps towards its realization in the *DSL-comet* tool. The tool permits creating and using mobile graphical DSLs on iOS devices, and their seamless use in desktop environments.

***Index Terms*—active DSL; graphical modelling language; flexible modelling; mobile application; API integration**

I. INTRODUCTION

Domain-specific languages (DSLs) are essential assets of software development paradigms like model-driven engineering (MDE) [1], and their use is also frequent in end-user development [2]. DSLs capture the main primitives within a domain – like production plant design, game development or logistics – and enable succinct and natural system descriptions using a terminology close to the domain experts. DSLs can be textual or graphical, but our focus is on graphical DSLs [1].

Modelling using DSLs has traditionally taken place in "static" environments, which typically run on desktop computers or laptops. In this vision paper, we claim that DSLs can greatly benefit from mobility and context [3]. We call these envisioned DSLs "*active DSLs*", and their role is to become central parts of collaborative, contextual mobile apps.

For example, imagine a DSL for smart city planning where the position of sensors within a city must be precisely determined; a DSL for tourism where tourist guides create touristic routes while walking through the city, and tourists can download the route models and rate their spots in-place while visiting the city; or a DSL for healthy living, where it is possible to create circuits for physical training, and some exercises can be proposed at specific locations as the circuit is created. Such DSLs would profit from their use on mobile devices, probably on-site, and the possibility to geolocate on a map the DSL elements (sensors, activities, touristic spots). The models built with the DSL may interact with external APIs upon user interaction or autonomously. For example, in the tourism DSL, routes might show the current temperature in the recommended spots, or display the entrance prices in different currencies. This information can be read from external services and can trigger model reorganizations. Moreover, some scenarios may require on-site collaboration, which should not rely solely on external WiFis or centralized servers, as users may be in remote locations with no connectivity.

We envision the rapid construction of interactive, collaborative, geo-enabled mobile apps, where active DSLs are central components. This approach has the potential to raise the impact of DSLs and MDE technologies in society beyond the current state, as nowadays the number of mobile users has surpassed the number of desktop users [4], and more than 40% of the world population is estimated to own a smartphone (peaking over 80% in some countries) [5]. Our proposal may also impact the way in which mobile industry works in the aforementioned scenarios, as mobile app builders would become language engineers, and mobile app users would become modellers. Hence, similar to the motto in [6], and paraphrasing Andy Warhol, we claim that *"(with active DSLs) in the future, everyone will be a modeller for 15 minutes"*.

This paper motivates the need for such a technology, and provides a classification of restricted classes of active DSLs. Moreover, we propose a supporting architecture, and show our first steps towards its realization in *DSL-comet*, a tool for mobile domain-specific modelling with incipient support for the concept of active DSL. The tool permits the creation and use of graphical DSLs on iOS mobile devices. It provides local collaboration facilities including a token-based model modification protocol, chats, informal drawings, and notes that can be geolocated and include pictures likely made with the device. The tool supports the usual mobile touch-based interaction gestures (e.g., swipe, tap, pinch) and integrates services like Dropbox, Google Drive or Twitter for model sharing. It also supports DSLs with elements geolocated in maps and information read from external APIs.

We presented an earlier version of *DSL-comet* in [3], [7], while in this paper, we propose the notions of active, geo, open and contextual DSLs; and introduce new features of the tool,

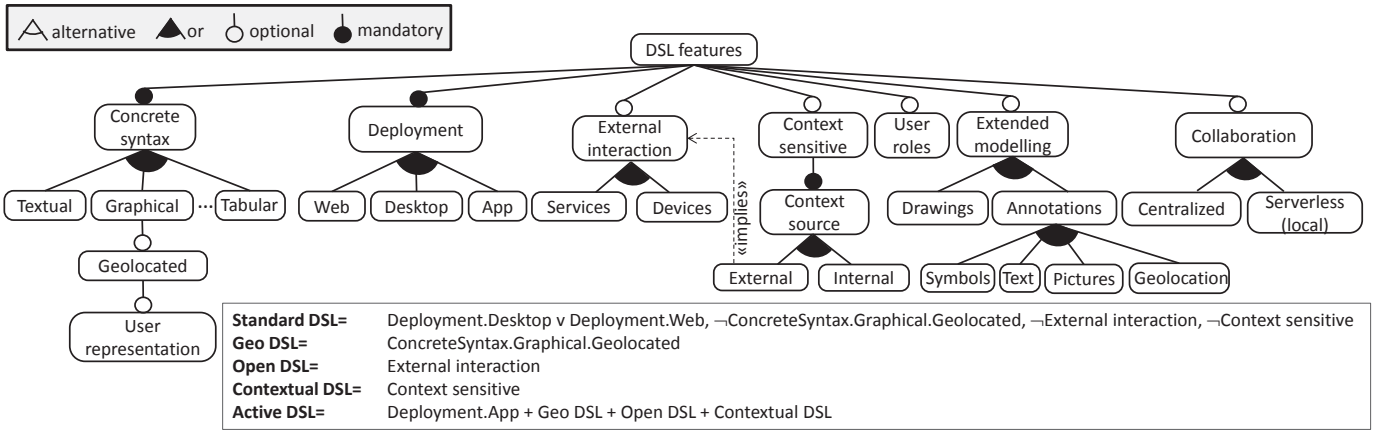

Fig. 1. Feature space for DSLs, and minimal features that characterise different types of DSLs

like geolocation of model elements, collaboration support and interaction with external systems.

In the rest of this paper, we first introduce some motivating scenarios in Section II. Then, in Section III, we elaborate on the concept of active DSL and its features. Section IV explains our approach to define active DSLs. Section V presents our tool and how it tackles the challenges posed by the scenarios. We discuss related works in Section VI, and draw some conclusions and lines of future work in Section VII.

II. MOTIVATION

Modelling in mobility provides flexibility of use in two aspects. First, modelling can take place in locations where using a computer would not be feasible. This permits being physically closer to the system being modelled (e.g., a wind turbine, a factory, the intelligent devices in a smart home) and incorporate information of the surrounding context in the models (e.g., geolocation or pictures taken with the mobile on the spot). Second, it promotes a more flexible collaboration among sets of users working or discussing on the same model, as they can physically move and arrange in ways which would be difficult to achieve with desktop computers or laptops.

As an example, consider a DSL to model home network configurations. Telecommunication engineers could use this DSL on their mobile devices to create configuration models on-site, while walking through each particular house. By adding suitable mechanisms, engineers could interact with the physical gadgets (e.g., changing their operation configuration) by manipulating the model elements, or show a partial view of the model with the gadgets in the current room.

Modelling with mobile devices can take advantage of the geolocation features of the device. For instance, in a DSL to design open-air circuits for physical exercises, a sports coach may be interested in inspecting the exercise locations. Hence, a system that allows creating the circuit in-place, over a map, would be helpful. In this setting, the placement of the objects on the map implies their geolocation.

Any of these scenarios would benefit from DSLs that facilitate the collaboration of nearby users. As the collaboration may take place in remote places, it cannot assume the availability of an Internet connection. Instead, it should rely on the short-range communication capabilities of the mobile devices, like Bluetooth or local WiFis. Other ways to enhance the flexibility of the collaboration include the support of informal drawings, markers, (geolocated) notes, and pictures likely made on-site with the mobile device.

Finally, to make the most of the models built in mobility, these should be processable by standard MDE tools like transformations and generators, and should be compatible with modelling editors on desktop computers.

III. CHARACTERIZING ACTIVE DSLs

In this section, we analyse the characteristics required to realize the full potential of active DSLs. We base the discussions on the feature model of Fig.1, which shows different features that can be incorporated into DSLs design, deployment and use. It also provides a classification of several types of DSLs according to their minimal features: *geo*, *open*, *contextual* and *active*. The latter type aggregates all features of the three former, and in addition, its deployment is as a mobile app.

The concrete syntax of DSLs can typically be graphical or textual, although other formats like tabular or form-based are also possible. Some DSLs may even define several concrete syntaxes of the same or different kind, which are used in combination. Active DSLs can have any kind of concrete syntax, but many scenarios we foresee require a graphical one to facilitate their use by end-users, and include the geolocation of some model elements in a map. Moreover, some scenarios may require representing the (possibly multiple) DSL users in the model, e.g., to identify their current location in a map, or to specify properties that can be attributed to them (e.g., nickname, energy or lives for DSLs in the active gaming domain). We call *geo DSLs* to DSLs that render their elements on an interactive map.

DSL editors can be deployed as desktop applications on computers, as web applications, or as mobile apps. Active DSLs require deployment as mobile apps to enable their use in mobility. Another required feature of active DSLs is the

capability to interact with external systems. This can be done via services (e.g., web services) or with devices like smart watches, smart bands or IoT devices. We call *open DSLs* to DSLs that communicate with external systems.

DSLs can be context-aware, in which case we call them *contextual DSLs*. Models built with this kind of languages can react to triggers caused by context elements taken either from the mobile device (like the remaining battery or current time) or from external sources (like web APIs). The latter implies interaction with external services, e.g., to obtain information about weather, traffic or pollution, among many others.

In some scenarios, the rendering, interaction and other features of a DSL may depend on the roles and permissions assigned to the DSL user. For example, a touristic DSL should let touristic guides create routes with spots of interest, while tourists should only be allowed to display route models and annotate their spots with evaluations or pictures (but not modify the route itself).

For flexibility, DSLs may enable extended modelling in the sense that they may accept the use of additional elements beyond those specified in the meta-model or grammar. These include text annotations, predefined symbols (e.g., indicating warnings, errors, the need to revise an item, etc.), pictures, geolocation information, or hand-made drawings.

Finally, collaboration support may be useful in many scenarios as well. We distinguish two kinds of collaboration: local and centralized. The former uses the short-range communication capabilities of the mobile device, does not require from connectivity, and can be used in remote locations (e.g., wind turbines, farms, the country side) or in the streets. The latter relies on a centralized server, and hence, it may coordinate collaborations from distant users.

Altogether, traditional DSLs are typically deployed as desktop or web applications, they do not have into account context or geolocation information, they generally lack external interaction, and only a few exceptions support user roles [8] or collaboration [9]. Instead, active DSLs are to be used in mobility and hence they are deployed as native apps on mobile devices. In order to unfold their full potential, they can use geolocation information, interact with external systems, be context-aware, support roles and enable collaboration.

After introducing the features required from active DSLs, the next sections describe our first steps towards its realization. First, Section IV presents our approach to define active DSLs, and then, Section V describes their usage on our tool.

IV. SPECIFYING AN ACTIVE DSL

Fig. 2 shows our approach to define geo, open, contextual and active DSLs. In all cases, the abstract syntax of the DSL is described through a standard domain meta-model. We use EMF in order to keep the compatibility with its extensive ecosystem. Then, this meta-model can be enriched with different annotation models to describe graphical representation, external interaction, geolocation and context rules (currently under development). The next subsections explain the annotation models we currently support and the deployment phase.

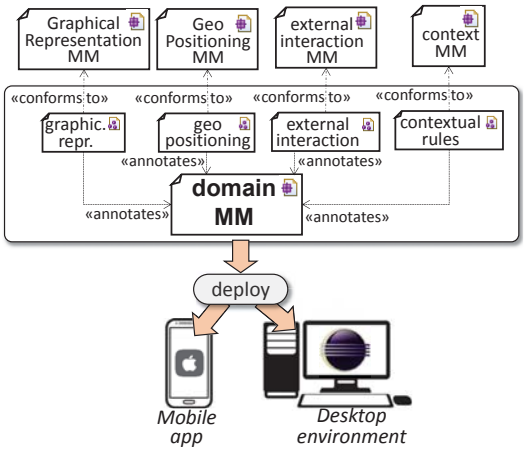

Fig. 2. The elements to describe and deploy active DSLs

A. Defining external interaction

We allow the value of attributes in a domain model to be retrieved from external sources via API calls. For this purpose, we have created the meta-model of Fig. 3, which permits describing how to perform calls to services and extract the data of interest from their result. We keep a repository of API call descriptions. Currently, we support web API calls with and without authentication, which return the data in JSON format. This can be easily extended in the future.

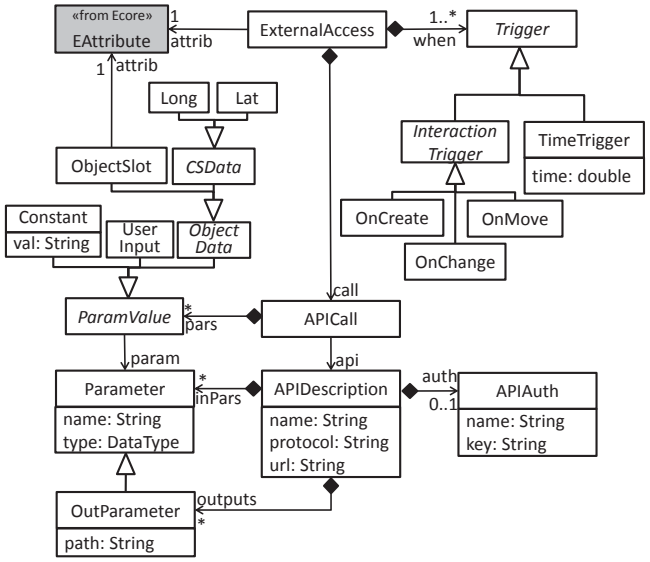

Fig. 3. Meta-model to describe external interaction (excerpt)

In order to configure an API call, it is necessary to specify how the values for the input parameters will be provided, how the value of the output parameters will be extracted from the returned data, and when to perform the API call. We currently consider four ways to set the value of input parameters: constant value, user input, another attribute of the object, or its geolocation. The latter is only possible for geo DSLs, as we will explain in Section IV-C. Regarding the output parameters,

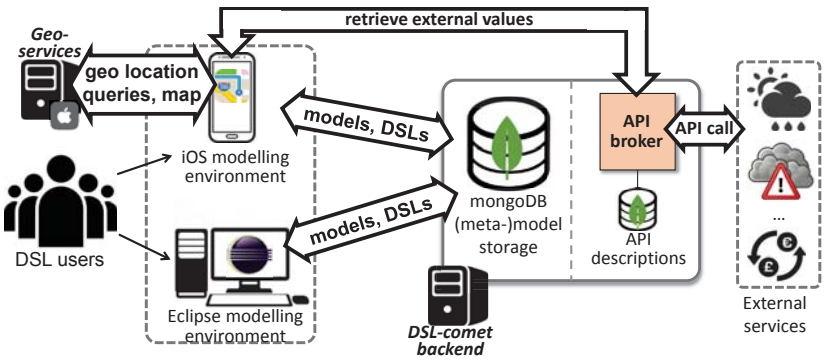

Fig. 4. *DSL-comet*'s architecture

they include an XPath-like expression [10] (attribute path) that indicates how to extract the parameter value from the returned JSON data. For example, the expression main.temp indicates that the value of the parameter is to be extracted from the temp field of the main dictionary. Finally, API calls can be triggered either in response to some user interaction (e.g., the object is created, moved or edited), or periodically.

B. Defining graphical representation

We also use annotation models to describe the graphical concrete syntax of the DSL. The goal is to have a concrete syntax compatible with EMF-based editors and deployable on mobiles. For this purpose, we have created a neutral meta-model called GraphicalRepresentation (presented in [7]) to describe the graphical visualization of each element of the DSL. The GraphicalRepresentation meta-model permits selecting and configuring the visualization of each class as a node or as an edge. In case of nodes, it is possible to attach them a geometrical shape or an icon, while for edges, it is possible to select the decorators for the start and end of the arrow.

Classes can also be tagged as Expandable. If so, their instances are not graphically represented in the model, but they are edited through a separate form that is presented when the container object is edited. If the container object is the canvas, then they can be edited through a special button. Expandable classes are useful in the reduced editing space of mobile devices to keep the model as simple as possible.

C. Defining geolocation

A last annotation model allows configuring whether the DSL will be geolocated, and whether the current position of the user is to be represented in the model. In such a case, the canvas will display an interactive map, and all visible model elements will be placed geolocated in the map. Internally, this is handled by extending the meta-model classes with two attributes lat and lon to store the latitude and longitude of each node in the map. The value of these attributes is automatically updated when the object is created or moved.

D. Deploying the DSL

The DSL definition (meta-model of its abstract syntax and annotation models) can be deployed in mobile devices using our tool *DSL-comet*, or they can be compiled to generate a desktop Eclipse editor based on the Sirius graphical modelling workbench [7]. However, technically, the features of geo and open DSLs are only available on mobile devices, and their inclusion in the generated Eclipse editor is future work.

In the following section, we describe how to use the defined DSLs in our *DSL-comet* tool.

V. DSL-COMET

This section presents our *DSL-comet* tool, aimed at supporting active DSLs with collaboration and extended modelling features. The tool runs on iOS devices, and can be installed from Apple's app store. Its home page is http://www.miso.es/tools/DSL-comet.html, and a video illustrating some of its features is available at https://youtu.be/rzhl9yMFSxI.

Next, we describe its architecture (Section V-A), basic features (Section V-B), collaboration facilities (Section V-C), and support for geo and open DSLs (Section V-D).

A. Architecture

Fig. 4 shows the scheme of *DSL-comet*'s architecture. In the front-end, the DSLs can be used both with Eclipse and iOS-based clients (iPhones and iPads). The meta-models, annotation models, and instance models are stored in a backend server. We use a mongoDB database for better scalability, whereby we have built services to convert between EMF and JSON. While the Eclipse client consumes models and meta-models in XMI format, the iOS client uses JSON.

Users can download DSL definitions from their mobile devices, or from an Eclipse client that is able to synthesize an Eclipse graphical editor based on Sirius [7], [11]. Then, users can use the DSL to build models, and store them either locally or in the server. The constructed models remain compatible with both the mobile and Eclipse-based editors.

For open DSLs, we have developed an API broker service in charge of two tasks. First, it holds API descriptions (conformant to the meta-model shown in Fig. 3) on a mongoDB database. Second, it is a mediator of API calls between the client and the service. Hence, when a mobile client needs to retrieve some data from an external service, it does so through the API broker. This results in a lighter client, a more flexible management of APIs, and facilitates the interaction with APIs

requiring a registration key, as this is done in the server instead of in each client. Currently, the broker service can only be used from the iOS client. Similarly, geo DSLs are only supported from the iOS client. They require internet connectivity for retrieving the map details and performing geolocation queries.

Next, we describe the main features of the iOS client.

B. Basic features

Fig. 5 shows the mobile client being used to create a model, using a DSL for describing home network configurations. First (label 1), the user can select DSLs stored either locally in the device or remotely in the server. Upon selecting one DSL (label 2), a retractable tool palette with the object types permitted by the DSL is loaded in the lower part of the tool (label 3). Objects are created by drag and drop of elements in the palette into the canvas (label 4), connections are built by long-pressing source and target objects, and object attributes are edited on a dialog box available after pressing over the object. The tool keeps the created model conformant to the meta-model by signalling connections with incompatible source/target objects, cardinality violations, and ill-typed attribute values.

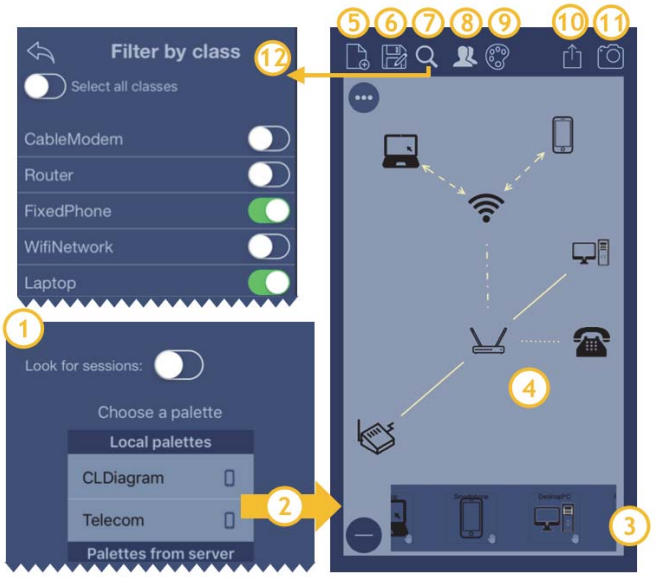

Fig. 5. Using *DSL-comet* to model home networks on an iPhone 7

Since mobile devices can have a reduced screen size (from 4.0 inches in an iPhone 5s, to 12.9 inches in an iPad Air), to save space, objects can be either created and displayed on the canvas, or be editable through dialogs but omitted from the canvas. This can be configured when defining the DSL, being the former the default behaviour, and annotating the classes whose objects will not be displayed as "Expandable".

On top of the screen, *DSL-comet* has buttons to create a new blank model (label 5), save the model locally or in the server (label 6), search model objects fulfilling certain criteria (label 7), look for collaboration peers (label 8, explained in Section V-C), select another DSL (label 9), share the model via external services (label 10), or take a snapshot of the model so that it can be shared via e-mail or Twitter (label 11). Label 12 shows the search facility, which supports filtering based on class types (in the figure) or attribute values.

Figs. 6 (a) and 6 (b) show the dialog boxes for editing node and edge attributes, which are presented upon tapping on a node or edge. They include appropriate controls to edit string, numeric, boolean or enumerate data types. The dialog for nodes (Fig. 6 (a)) also shows the incoming and outgoing edges. The dialog for edges (Fig. 6 (b)) shows the source and target classes, as well as the edge attributes, if any.

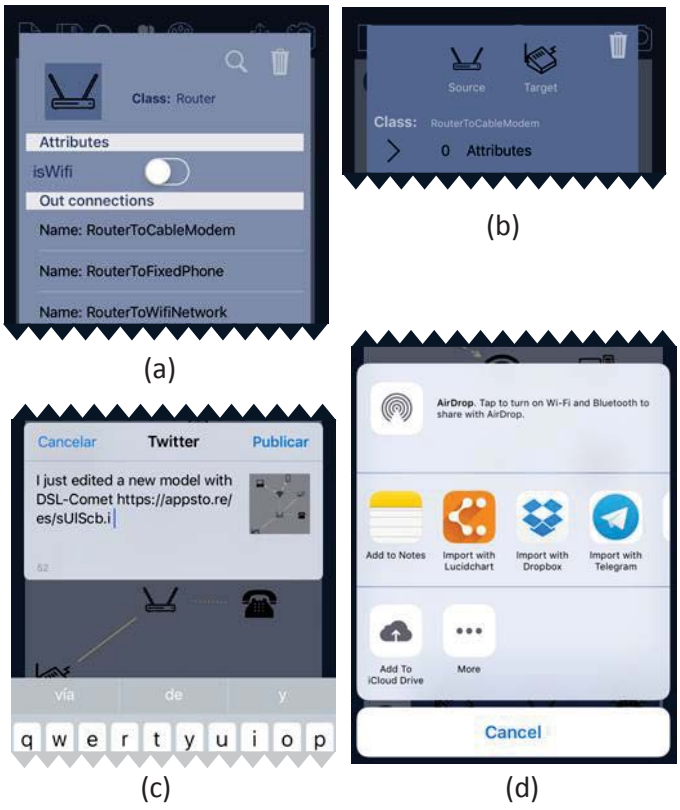

Fig. 6. Editing nodes (a) and edge details (b). Sharing models through Twitter (c), Dropbox, Telegram, Drive and others (d).

Figs. 6 (c) and 6 (d) show some model sharing options via Twitter, Air Drop and other applications installed in the mobile device, like Dropbox, Telegram, iCloud and Google Drive. In the case of Twitter, it is possible to share an image of the model, while in the other cases, the model source is shared.

C. Collaboration features

The tool supports collaborative modelling. For this purpose, first one of the devices must initiate the collaboration (the so-called "server"). This device sets up a local WiFi to which nearby devices can connect. The collaboration is token-based, where only the user holding the token is the "master" and can modify the model (i.e., create, delete, modify or move objects in the canvas). Any user can request the modification token, which the server can grant or deny, whereas the server can get the token at any moment. Model changes are distributed

at once to the rest of devices in the collaborative session. As an example, Fig. 7 shows a moment in a collaboration session between two devices: an iPad mini (at the back) and an iPhone SE (at the top). The iPhone started the collaboration and has the capacity to expel or promote users into masters.

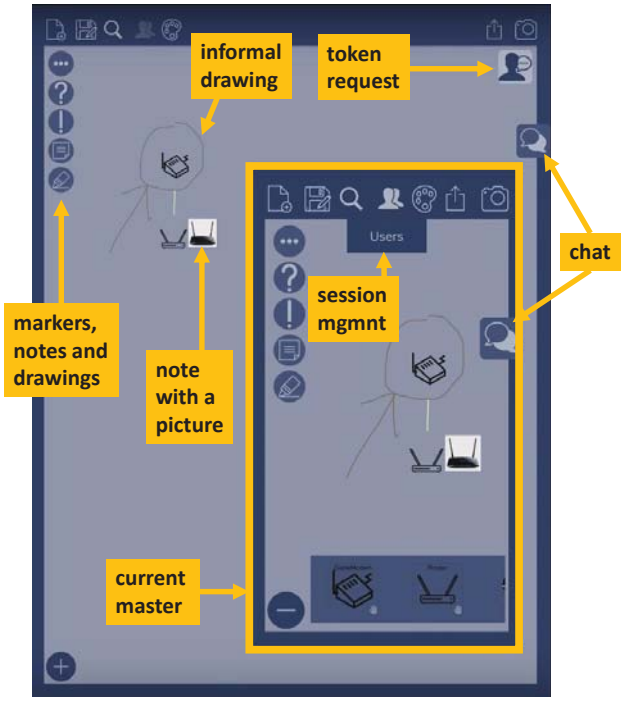

Fig. 7. A collaboration session with two devices

To provide agility to the collaboration, the tool supports extended modelling by providing some useful elements that can be used independently of having the modification token or not: temporal markers, notes, freehand drawings and a chat. Temporal markers ('?' and '!') permit directing the collaborators' attention to some part of the model, and after a few seconds, the marker disappears. Notes are permanent and can be attached to objects or be placed anywhere in the canvas. A note can have text, be geolocated in a map, and include a picture made on the spot or selected from the camera roll. Users can also draft freehand drawings and delete them once they are not needed anymore. As an example, the model in Fig. 7 has a note and a freehand drawing. The note attaches a picture of a router, which in this context is useful to document the real device the model object refers to. The informal drawing has been sketched by one of the users in order to highlight some object in the model, similarly to what one would have done using pen and paper.

D. Open and geo DSLs

DSL-comet supports geo DSLs where model elements are displayed on a map and become geolocated. For this purpose, the meta-model of the DSL must be annotated as "Geolocated", in which case, all its visible elements are implicitly added attributes to store their geolocation. It is also possible to specify API calls from which attributes will receive their values. Although open and geo DSLs can be used for modelling both in mobile and Eclipse-based desktop editors, only the former devices are able to fully exploit the geolocation services, perform API calls and automatically assign a geolocation to objects based on their location in maps.

Fig. 8 shows an open, geo DSL for tourism. Tourism agencies can use it to create models of touristic routes that include monuments, museums and restaurants. Tourists can download the models to follow the routes, comment or rate the different spots, and attach and share pictures taken in-place.

Fig. 8. Open, geo DSL for touristic routes

The model in the figure corresponds to a route through the city of Madrid, whose map is displayed in the canvas (label 1). The nodes represent tourist attractions and are geolocated (label 2). The displayed map is configurable (satellite, hybrid, etc.). The tool supports Google-like queries using natural language (label 3), and when they return an existing place (e.g., "Museo del Prado"), the map gets centred on that place. The current location of the user is shown in the map (label 4), who can attach pictures to the different model elements. Moreover, the local collaboration facilities of the tool permit groups of tourists to share the attached pictures on the model, add temporal markers (like the exclamation mark in the figure) to signal interesting attractions, and use the chat. Each touristic spot has an attribute `temperature` that is read from a call to `openweathermap`'s API. The value is refreshed when the object is created, edited or moved.

Overall, we believe this new kind of DSLs can be the basis for highly interactive, collaborative and geo-enabled apps.

VI. RELATED WORK

There are several applications in the market, and a few academic ones, to create drawings, diagrams or models in mobile devices. Table I shows a summary of them.

Lucidchart [12] is a commercial diagramming web-based tool, which also has a mobile version. It features several palettes which can be used in combination, as the tool does not

TABLE I
COMPARISON OF MOBILE DIAGRAMMING TOOLS

Tool	Type	Palette	Collab.	Geo-serv.	Extra
Commercial tools					
Lucidchart [12]	diagr.	yes	server web	no	
DrawExpress [13]	sketch	yes	no	no	pictures
SyncSpace [14]	drawing	no	local	map pics.	pictures
Ideament [15]	diagr.	graphs	no	no	pictures
Academic tools					
CEL [16]	UML	class diag	no	no	
FlexiSketch [17]	reqs	no	yes	no	
Pounamu [18]	DSLs	yes	no	no	
DSL-comet	DSLs	yes	local	maps search geoloc.	pictures drawings protocol API access

rely on meta-models or performs semantic checks. It supports collaboration between mobile and web users, but there is no collaboration protocol, which may cause collisions, and communication is done through a central server, which may cause delays. DrawExpress [13] features both a palette and sketch recognition of simple shapes like circles or rectangles. SyncSpace [14] is a drawing tool (i.e., there are no diagrams, just lines) which supports local collaboration, similar to our approach. Drawings can be done on a map, though this map is first converted to an image, so no geolocation services like searches are possible. Ideament [15] can create graphs, but does not offer palettes (just some shapes) or collaboration.

Some apps have been created for a particular modelling language. CEL [16] is a mobile app to create UML class diagrams with no support for collaboration or model sharing, and Puzzle [19] permits creating mobile apps using graphical modelling in the device. In these approaches, the languages are fixed, and the apps were created ad-hoc for them. Instead, we enable the use of arbitrary DSLs, and the app is configured with the DSL descriptions. Hence, our approach could help creating this kind of applications.

FlexiSketch [17] is a sketching mobile modelling tool tailored for software requirements modelling. Diagram nodes can be either images or user sketches which get converted into entities that can be manipulated and assigned a type. The tool supports collaboration, but no protocols or geo-services.

Although MDE has been used to produce mobile applications [20], few works report on using DSLs on mobiles. One of the few is Pounamu/mobile [18], an extension of the Pounamu meta-tool to generate mobile user interfaces to interact with a DSL. However, every interaction goes through a server, with the consequent delay. Taking into account that the work is from 2006, the tool offered no collaboration or geo-services.

Some tools allow graphical modelling in the web [21], [22], [23]. They can be used from a mobile device using a web browser, but this poses some drawbacks. First, web applications are not tailored to the particularities of mobile devices, while apps are optimized for them, enabling forms of visualization and interaction designed for the reduced screen size. Second, web applications require connectivity, which might not be available in remote locations. Finally, relying on a web application for collaborative modelling might involve greater delays than the local form of collaboration we support.

Some researchers have proposed modelling the user experience (UX) of modelling tools [24], and much work exists on context modelling and reasoning techniques [25], [26]. For the full realization of active DSLs, we will adapt these techniques to work with DSLs.

Altogether, our proposal is new as none of the previous tools permits modelling (in contrast to drawing) using geolocated DSLs, permits defining access to external APIs, or enables short-range collaboration for modelling. Moreover, the very concept of active DSL is novel as well.

VII. CONCLUSIONS AND RESEARCH ROADMAP

In this paper we have introduced the notions of geo, open, contextual and active DSLs. Open DSLs can interact with external systems. Geo DSLs render the models over a map profiting from geo-services. Contextual DSLs are context-aware and can reorganize a model upon context triggers. The features of these kinds of DSLs can be combined into an active DSL, which in addition is deployed in a mobile. We have presented the first steps towards realizing active DSLs in the *DSL-comet* tool. Its architecture permits creating DSLs for both Eclipse and iOS mobile devices. The mobile client supports local collaboration, the use of geolocation services within DSLs, and the interaction with external services.

Active DSLs open the use of MDE to a new range of applications, including smart interfaces for IoT applications, and we urge the community to explore this topic. A research roadmap to move from tools for standard DSLs towards a full realization of active DSLs implies the support for open and geo DSLs first, and then for contextual capabilities as described in Section III. Modelling in mobile devices is central to this vision, and implies diverse challenges such as a smart use of the reduced screen size and lighter model persistence formats than standard ones based on EMF. While we have devised light persistence mechanisms based on JSON, standardization of this aspect would be useful. Similarly, we have developed our own API description mechanisms (cf. Fig. 3), though standardization would facilitate the construction of open DSLs [27]. Further challenges include the exploitation of crowd modelling, and the combination of local and global synchronization mechanisms.

Regarding *DSL-comet*, our immediate plans include the exploration of context rules for model adaptation. We will also exploit the possibilities of geo DSLs by permitting several users to share their location and appear on the same map, enable the definition of user roles, and explore the creation of domain-specific mobile apps based on active DSLs.

ACKNOWLEDGMENT

Work funded by the Spanish MINECO (TIN2014-52129-R) and the R&D programme of Madrid (S2013/ICE-3006).

REFERENCES

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

[2] A. J. Ko, R. Abraham, L. Beckwith, A. F. Blackwell, M. M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. A. Myers, M. B. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck, "The state of the art in end-user software engineering," *ACM Comput. Surv.*, vol. 43, no. 3, pp. 21:1–21:44, 2011.

[3] D. Vaquero-Melchor, A. Garmendia, E. Guerra, and J. de Lara, "Towards enabling mobile domain-specific modelling," in *ICSOFT*. SciTePress, 2016, pp. 117–122.

[4] D. Chaffey, "Mobile marketing statistics compilation," http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/, SmartInsights, Tech. Rep., 2017.

[5] Y. Wu, "Global smartphone user penetration forecast by 88 countries: 2007 - 2022," Strategy Analytics, Tech. Rep., 2016.

[6] F. Hermans, J. Siegmund, T. Fritz, G. Bavota, M. Nagappan, A. Hindle, Y. Kamei, A. Mesbah, and B. Adams, "Leaders of tomorrow on the future of software engineering: A roundtable," *IEEE Software*, vol. 33, no. 2, pp. 99–104, 2016.

[7] D. Vaquero-Melchor, A. Garmendia, E. Guerra, and J. de Lara, "Domain-specific modelling using mobile devices," in *Selected papers from ICSOFT 2016. CCIS 713*. Springer, 2017, pp. 221–238.

[8] G. Bergmann, C. Debreceni, I. Ráth, and D. Varró, "Query-based access control for secure collaborative modeling using bidirectional transformations," in *Proc. MODELS*. ACM, 2016, pp. 351–361.

[9] J. Gallardo, C. Bravo, and M. A. Redondo, "A model-driven development method for collaborative modeling tools," *J. Network and Computer Applications*, vol. 35, no. 3, pp. 1086–1105, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2011.12.009

[10] W3C, "XML Path Language (XPath) 3.1," https://www.w3.org/TR/xpath-31/, 2016.

[11] Sirius, 2016. [Online]. Available: https://eclipse.org/sirius

[12] Lucidchart, 2016. [Online]. Available: https://www.lucidchart.com

[13] DrawExpress, 2016. [Online]. Available: http://www.drawexpress.com

[14] SyncSpace, 2016. [Online]. Available: https://infinitekind.com/syncspace

[15] Ideament, 2016. [Online]. Available: http://www.nosleep.net

[16] R. Lemma, M. Lanza, and F. Olivero, "CEL: modeling everywhere," in *ICSE*. IEEE / ACM, 2013, pp. 1323–1326.

[17] D. Wüest, N. Seyff, and M. Glinz, "Flexisketch: A mobile sketching tool for software modeling," in *MobiCASE*, ser. LNICST, vol. 110. Springer, 2013, pp. 225–244.

[18] D. Zhao, J. C. Grundy, and J. G. Hosking, "Generating mobile device user interfaces for diagram-based modelling tools," in *AUIC*, ser. CRPIT, vol. 50. Australian Computer Society, 2006, pp. 101–108.

[19] J. Danado and F. Paternò, "Puzzle: A mobile application development environment using a jigsaw metaphor," *J. Vis. Lang. Comput.*, vol. 25, no. 4, pp. 297–315, 2014.

[20] E. Umuhoza and M. Brambilla, "Model driven development approaches for mobile applications: A survey," in *MobiWIS*, ser. LNCS, vol. 9847. Springer, 2016, pp. 93–107.

[21] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. V. Mierlo, and H. Ergin, "AToMPM: A web-based modeling environment," in *Demos @ MoDELS*, vol. 1115. CEUR, 2013, pp. 21–25.

[22] M. Maróti, T. Kecskés, R. Kereskényi, B. Broll, P. Völgyesi, L. Jurácz, T. Levendovszky, and Á. Lédeczi, "Next generation (meta)modeling: Web- and cloud-based collaborative tool infrastructure," in *MPM @ MoDELS*, vol. 1237. CEUR, 2014, pp. 41–60.

[23] L. M. Rose, D. S. Kolovos, and R. F. Paige, "EuGENia live: A flexible graphical modelling tool," in *XM @ MoDELS*. ACM, 2012, pp. 15–20.

[24] V. Sousa and E. Syriani, "An expeditious approach to modeling IDE interaction design," in *Proc. GEMOC+MPM@MoDELS*, ser. CEUR Workshop Proceedings, vol. 1511. CEUR-WS.org, 2015, pp. 52–61.

[25] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010.

[26] S. Ceri, F. Daniel, F. M. Facca, and M. Matera, "Model-driven engineering of active context-awareness," *World Wide Web*, vol. 10, no. 4, pp. 387–413, 2007.

[27] OpenAPI, "Open API initiative," https://www.openapis.org, 2017.