# Towards access control for collaborative modelling apps

Léa Brunschwig
Universidad Autónoma de Madrid
Madrid, Spain
lea.brunschwig@uam.es

Esther Guerra
Universidad Autónoma de Madrid
Madrid, Spain
esther.guerra@uam.es

Juan de Lara
Universidad Autónoma de Madrid
Madrid, Spain
juan.delara@uam.es

## ABSTRACT

Domain-specific languages (DSLs) are small languages tailored to narrow domains. Their purpose is to cope with the needs of domain experts, who might not have a software engineering background. In previous work, we proposed the novel notion of Active DSLs, which are graphical DSLs extended to benefit from mobility using geolocation and interactions with external services and devices. Active DSLs are the central component of a mobile collaborative appl called DSL-comet.

Modelling using DSLs can be done collaboratively by a group of stakeholders, and the levels of required confidentiality and integrity may vary across modelling artefacts. While preventing the access to protected data has been tackled for DSLs used on static environments like laptops and desktop computers, it has not been envisioned for modelling on mobile devices. The latter poses further challenges as access permissions may depend not just on user profiles but also on conditions that only make sense in mobility, such as geolocation or information retrieved from nearby sensors.

Embracing the approach of Active DSLs, we propose an annotation meta-model to provide fine-grained role-based access control to any domain meta-model, hence enabling model element protection when collaborating in mobility. The paper describes our current implementation and our envisioned low-code solution, which includes a cloud-based textual editor to define role hierarchies and permissions for the domain meta-models.

## CCS CONCEPTS

• **Software and its engineering → Domain specific languages**.

## KEYWORDS

Mobile modelling, Active DSLs, Role-based access control, Collaborative modelling, Mobile app, Low-code engineering platforms

**ACM Reference Format:**
Léa Brunschwig, Esther Guerra, and Juan de Lara. 2020. Towards access control for collaborative modelling apps. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3417990.3420201


*MODELS '20 Companion, October 18–23, 2020, Virtual Event, Canada*

ACM ISBN 978-1-4503-8135-2/20/10…$15.00
https://doi.org/10.1145/3417990.3420201

## 1 INTRODUCTION

Modelling is a collaborative activity that consists of capturing the essence of a system for a specific purpose. This task is part of a large variety of disciplines in engineering, the natural and the social sciences, and serves activities like system construction, maintenance, simulation, analysis or testing.

In the context of software engineering, model-driven engineering (MDE) places models in the centre of software development [6, 22]. Models are often defined with domain-specific languages (DSLs) [14, 17, 35]. Unlike general-purpose languages, DSLs are shaped according to a concrete domain. Since they focus on the domain area of the application and not on its technical details, end-users can better use and understand DSLs [2, 15].

Traditionally, modelling begins with pen and paper or whiteboards. When models need to be machine-processable, e.g., for analysis or code generation, modelling occurs on static environments such as desktop computers or laptops, assisted by modelling tools like the Eclipse Modeling Framework (EMF) [7]. However, according to recent studies [31, 37], the number of mobile users is greater than the number of desktop users, with an estimation of around 45% of the population owning a smartphone in 2020 (around 3 500 000 000 smartphone users). Moreover, the increasing capabilities of mobile devices, with a wide variety of embedded sensors and great computing power, is another reason that makes the idea of modelling on mobile devices strongly appealing. Finally, modelling on mobile devices can bring modellers closer to the system being modelled, and simplify the specification of some domain requirements, such as geolocation data [23]. Otherwise, these models may need to be defined with pen and paper on-place and then transferred into a computer at the working place. For these reasons, several tools have emerged to tackle this challenge, like NetSketcher [1], CEL [16], HoloFlow [24] or DSL-comet [32].

Models are used at every stage of development, and it is, therefore, natural to accomplish their design in collaboration [12]. We distinguish between *asynchronous collaboration*, which is done individually by working on a model shared on a common repository; and *live collaboration*, which can be done remotely [18] or in the same location by sharing either the network [38] or the screen [10]. However, access to some models may need to be restricted for some users, and some stakeholders may have different model editing privileges. For instance, car systems are critical systems that should not be seen or edited by anyone due to the risks for human lives. To cope with this challenge, we propose applying role-based access control for modelling on mobile devices [20]. Access control for models used on static environments like laptops and desktop computers, has been tackled by other researchers [4], but it has not been envisioned for modelling on mobile devices. This poses further challenges as access permissions may depend not just on user

profiles but also on conditions that only make sense in mobility, such as geolocation or information retrieved from nearby sensors.

In this paper, we present a meta-model for role-based access control for Active DSLs [34] that takes into account mobility aspects. Active DSLs are graphical DSLs that run on mobile devices. They are defined by a domain meta-model enriched with additional meta-models that define aspects of graphical syntax, external interactions or geo-positioning. We show the first steps towards the realization of our approach in DSL-comet, a tool for mobile domain-specific modelling presented in [32–34]. We also report on our vision of a low-code development environment for creating Active DSLs, including their role-based access control.

The rest of this paper is structured as follows. We first motivate our work in Section 2 and provide background in Section 3. We introduce our User Role DSL and envisioned architecture in Section 4. Section 5 reports on tool support. Finally, we compare our approach with related works in Section 6, and draw some conclusions and lines of future work in Section 7.

## 2 MOTIVATION

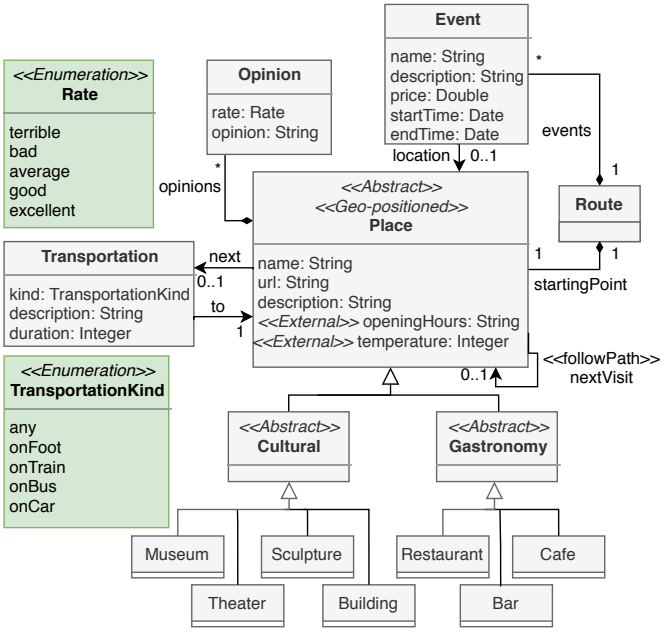



**Figure 1: Tourism meta-model.**

To motivate the need of control access in collaborative modelling apps, we will use an example scenario concerned with a DSL for tourism that involves different categories of stakeholders. The tourism DSL of Figure 1 would be used by both touristic guides and tourists. The former will create route models with spots of interest, while the latter should only be allowed to display the routes. Hence, we can underline the need for roles and permissions to control the actions of users depending on their identity.

Modelling on mobile devices is relevant in this scenario as it brings upfront the flexibility provided by mobility. The touristic guides will be enabled to model in locations where desktop computers cannot be used, such as the streets of a city. Moreover, they will be close to the route being modelled, and so, they will be able to incorporate information of the surrounding context in the model: the geolocation of points of interest and their pictures, and the modelling on a map (instead of in a blank canvas).

While making a guided tour, guides could also initiate a local collaboration session with the tourists via Bluetooth or public Wi-Fi of the city. This would permit tourists to access additional information (e.g., opening hours of museums, entrance fees), to situate themselves on the map to know the time remaining for the tour, and to give feedback on the route by adding notes or pictures taken on-site. This collaboration could also be broadcast to allow latecomers to join the guided tour by locating the guide. This collaboration could be moderated using roles and permissions to decide who can access these pieces of information (e.g., only the tourists who paid for the tour) and to decide whether tourists can provide feedback.

Finally, to make the most of this scenario, the defined routes should be processable by MDE tools and compatible with modelling editors for desktop computers. This way, touristic guides could edit the routes back at the office, tourists could select in advance the touristic routes of interest on their desktop computer at home, and model transformations/code generators could be used to generate popularity reports of the routes.

In this paper, we aim to facilitate the definition of access control policies for collaborative modelling languages that are to be used in mobility, such as the tourism DSL. For this purpose, we propose a novel DSL to configure the different user roles (e.g., touristic guide, tourist) and their permitted actions (e.g., creation of certain types of objects, adding pictures to the models). These actions may depend on contextual conditions which arise when modelling occurs in mobility (e.g., proximity to a given geoposition). The design of this role DSL is driven by the following requirements:

**R1:** Permissions should be reusable among roles;
**R2:** Permissions should be applicable on elements of the DSL;
**R3:** Permissions should be applicable on editor functionalities;
**R4:** Permissions may depend on contextual conditions;
**R5:** Users can request a role.

To tackle R1 we organize roles in a hierarchy (similar to inheritance in object oriented proramming) so the permissions can be reused between them. This way, a touristic guide can inherit all permissions given to tourists. R2 states that the permission can be applied on elements of the DSL, for example stating that tourists cannot add new *Place* or change the attribute *rate* of class *Opinion* of Figure 1. Besides, we can consider restricting the functionality of the mobile editor like the collaboration functionality mentioned earlier (R3). R4 says that permissions can be also active in specific conditions depending on the context, for instance, tourists can only add pictures of a place if they are less than 50 meters away from it. This requirement reinforces the argument stating that modelling on mobile devices has advantages compared to static modelling. Finally, users who do not have a role yet, or want to change their role, would be allowed to request a new one (R5).

# 3 BACKGROUND

Before diving into the core of our solution, we provide some necessary background. In the following, we first introduce Active DSLs, and then we give an overview of our tool DSL-comet.

## 3.1 Active DSLs

Generally, DSLs are deployed on desktop or web applications and do not consider geolocation information or context. They tend to lack external interaction (e.g., to enable attribute values in objects be read from external services or sensors) and only a few of them make use of collaboration [13] or user roles [3]. Yet, the motivating example of Section 2 proves that such kinds of features are needed in some scenarios. In the tourism DSL, the touristic guide would like to geolocate some model elements (e.g., touristic spots), connect them to external web services for retrieving their opening hours, and initiate a collaborating session with tourists while doing the tour in order to have live access to their opinions. On their side, tourists should only be able to query the routes and share their pictures and opinions. Considering this example, the tourism DSL has to be used in mobility because the model itself is the core functionality of the app.

Active DSLs are a novel approach, proposed in [34], which can respond to the touristic guide's needs. Figure 2 illustrates the elements to describe and deploy Active DSLs.

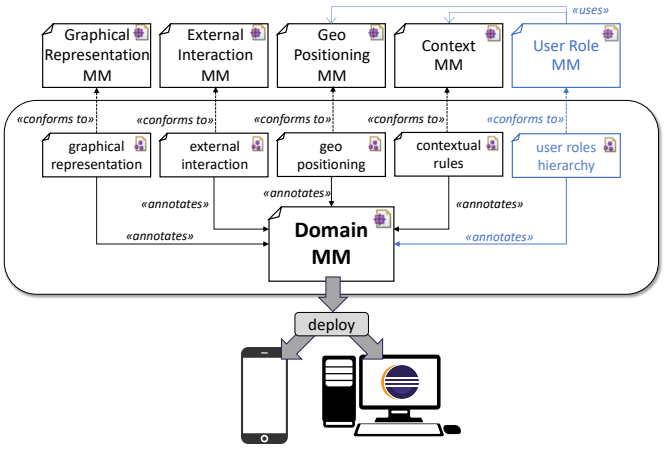



**Figure 2: Elements to describe and deploy Active DSLs.**

Active DSLs can be deployed on both mobile devices and desktop computers, though their deployment on mobile apps permits exploiting some features which are only relevant in mobility (e.g., geopositioning). An Active DSL is defined by a domain meta-model, plus several annotation models that configure different concerns of the DSL. The annotation models are conformant to the meta-models shown in the upper part of Figure 2, which we describe next.

First, Active DSLs can have any concrete syntax. Nevertheless, we advocate the use of a graphical syntax to represent the instances of the domain meta-models, in order to facilitate their use by end-users.

Active DSLs can also profit from geolocation of some model elements in a map, and represent the DSL users in the model to identify their current position thanks to a geo-positioning meta-model. In Figure 1, we annotated class *Place* as geopositioned, and the *nextVisit* edge as *followPath*, so that these links are displayed following the paths on the map (e.g., through the streets of the city).

Another characteristic of Active DSLs is the ability to communicate with external systems. The external interaction meta-model considers two kinds of external elements: external devices (e.g., IoT devices) and services (e.g., web services). In our example in Figure 1, the *openingHours* of the place and the *temperature* are obtained from external services.

The context meta-model permits defining contextual rules to render the domain meta-model context-aware. This permits an Active DSL to react to external events triggered by external interactions (like a web API and IoT devices). Similarly, they can react to the current state of built-in functionalities of the mobile device (such as sensors or internal clock).

Finally, in certain circumstances, the rendering of the model or the user interaction may depend on the roles and permissions granted to the users. For example, the user role meta-model would serve to implement the access rules described for the tourism DSL. Section 4 will present our proposal for such a user role meta-model, which is the core contribution of this paper.

In addition to these aspects of an Active DSL, specified as annotation models, many modelling scenarios also require collaboration. There are two forms of collaboration: serverless (also called local) and centralized. The former uses short-range communication capabilities of the hosting mobile devices. Its advantage is the aptness to be used in remote locations (e.g., countryside, wind turbines) or the streets. The latter relies on centralized servers and permits remote collaboration between distant users. In addition, Active DSLs can enable extended modelling, which refers to the use of additional elements beyond those described in the language. Text annotations, pictures or sketches are extended modelling capabilities that can enrich the collaboration or the semantics of a model.

Overall, Active DSLs are meant to be used on mobile devices to take advantage of their full potential. Their configuration is given by a set of annotation models that enrich a domain meta-model. They can use geolocation data, benefit from context-awareness, interact with external systems, support roles and enable collaboration.

## 3.2 A brief overview of DSL-comet

DSL-comet is a tool aimed at supporting Active DSLs. The tool runs on iOS devices and can be installed from Apple's app store. Its home page is https://diagrameditorserver.herokuapp.com, and a demonstration video to illustrate some of its features is available at https://youtu.be/rzhl9yMFSxI.

Figure 3 shows the architecture of the tool. Active DSLs in DSL-comet can be used both within Eclipse and iOS-based clients (iPhones and iPads). The domain meta-models, their instances and their annotation models are stored in a mongoDB database for scalability. The iOS modelling environment uses models and meta-models defined with JSON, and the Eclipse client consumes XMI format. A backend server provides services to convert between EMF and JSON in order to maintain the compatibility between the two clients.

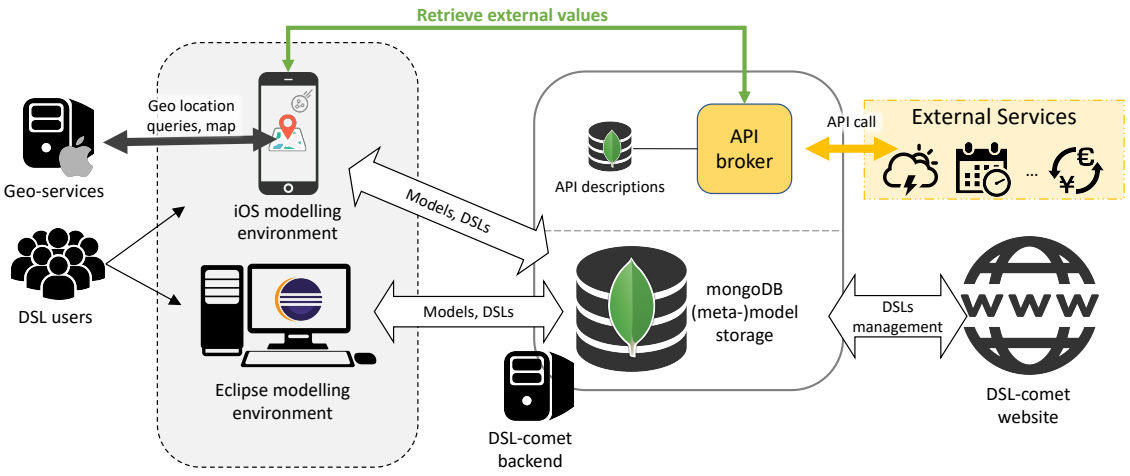



**Figure 3: Architecture of DSL-comet.**

Users can download the Active DSL definitions stored on the database from mobile devices or from an Eclipse client that uses Sirius [29] to provide a graphical syntax to the DSLs. Next, users can start building models and store them either locally or in the database. In addition, the website of DSL-comet permits the management of all meta-models and models stored in the database, namely, adding new meta-models and palettes (graphical syntaxes) and consulting the list of stored artefacts.

DSL-comet provides an API broker service to manage the interaction with external services. Therefore, when a mobile client needs to retrieve data from an external service, it does so through the API broker. This facilitates the interaction with APIs requiring a registration key since this is done by the server and not by the client.

The last feature of DSL-comet is the support of geo-services. Specifically, when the mobile client is connected to the internet, it can use those services for retrieving the map and performing geolocation queries. This allows for creating geo-located models.

Figure 4 shows a screenshot of a geo-located model, which we are using to enumerate the basic features of the DSL-comet modelling environment. First (label 1), the user can drag and drop language elements from the palette to the canvas (label 2). Once positioned (label 3), the item gets geolocated and will remain placed accordingly on the map. It is also possible to draw references between items if the syntax of the DSL allows it. If the model is geolocated, then the references can be configured to follow routes on the map (label 4).

Users can also be represented with a geolocated custom image on the map at their current location (label 5), and their attributes can be edited as illustrated in Figure 8.

In Figure 4, the top of the screen displays a menu with buttons to create a new blank model (label 6), save the model either locally or on the server (label 7), search model objects (label 8), look for peers to initiate a collaboration session using Bluetooth (label 9), select another DSL (label 10), share the model source via external services (label 11), or take a screenshot of the model so it can be shared (label 12). Model sharing can be done via services like e-mail and AirDrop, or using other applications installed on the device such as Twitter, Telegram, DropBox or Google Drive.


**Figure 4: Screenshot of a geo-located model in DSL-comet.**

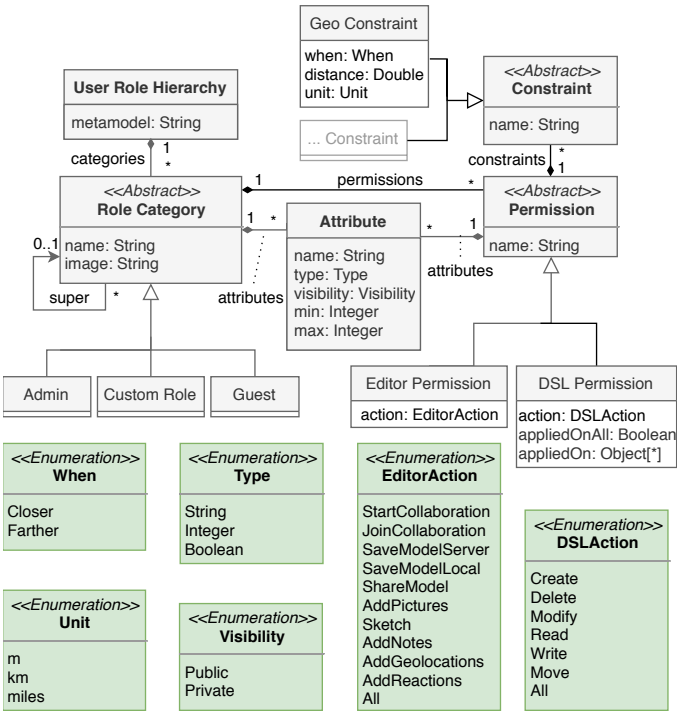
In the remainder of the paper, we introduce the user role DSL that we have designed to allow defining role-based access control policies for Active DSLs, as well as the first steps towards its realization atop DSL-comet.

## 4 USER ROLE DSL

Some scenarios need to secure access to some part of the models or to control their editing. For that purpose, this section first describes a new meta-model for defining roles and permissions to add to Active DSLs, and then it describes its concrete syntax.

### 4.1 Abstract syntax

We have created the meta-model of Figure 5, which permits describing role hierarchies and permissions to restrict the access and use of an Active DSL and its instances.




```
context UserRoleHierarchy
inv:
   self.categories -> one(c | c.oclIsTypeOf(Admin)) and
   self.categories.oclIsTypeOf(Guest).size() = 1

context Admin
inv: self.permissions -> size() = 0

context DSLPermission
inv:
   self.appliedOnAll implies (self.appliedOn -> isEmpty())
```

**Figure 5: User role meta-model.**

*Roles are organized in a hierarchy* (R1) and each of them has an optional image that can be used to distinguish different roles during collaboration in addition to the name, and zero to many permissions. We distinguish the following three kinds of roles:

- *Admin* which refers to the author of the language and the administrators of DSL-comet (we will get back to them in Section 5). Every Active DSL must include exactly one role of this kind, which gives access to all functionalities and elements of the language.
- *Guest* which refers to users who have not been assigned any other role or are not logged in (see Section 5). This is typically the role with the least permissions, and by default, it forbids access to the model. Hence, if a user role model does not declare any other role (apart from the Admin) and no permissions are granted to the guest role, then no one except the Admins will be able to use and create instances of the language. There is exactly one role of this type.
- *Custom Role* which refers to language-specific roles, such as touristic guides and tourists.

Likewise, permissions are classified into two distinct groups which determine their action scope:

- *Editor permissions* are *applied on functionalities of the editor* (R3) e.g., allowing collaboration, model sharing, attaching pictures or reactions (visual alerts) to model elements, among others. These actions currently refer to functionalities of DSL-comet, but could be applicable to other editors as well.
- *DSL permissions* concern the management (i.e., creation, deletion, modification, etc) of the *elements of the domain meta-model*. They can target classes as well as their attributes and references (R2).

The two permissions have their own specific actions but share the action "all" which implies that it is allowed to perform all the other actions.

Regarding the DSL permissions, the boolean "appliedOnAll" permits selecting all the classes and their element to apply an action on them. We support the classical CRUD actions on the abstract syntax (create, delete, modify and read). In addition, we support actions on the concrete syntax, like move, which refers to the possibility to change the position of an element on the canvas. More especially, for geolocated elements it allows (or not) to change their geopositioning information. In our example, a touristic guide should have the right to move all the elements but a tourist must not be able to change the location of a museum on the map. Finally, it is important to understand the difference between write and modify: the first one allows to fill the values of the attributes but it cannot be changed afterwards while the latter one implies that it can be changed when so desired.

Permissions and role categories can define attributes to be informed when the language is deployed. As an example of the former, one could define the address or the name of an external service or device, to restrict its exploitation by users. As an example of the latter, one could define that tourists have a name and age, and touristic guides speak a number of languages. The value of this attribute should be informed for each particular user having the role. Attributes so defined can be either public or private, which mean a public attribute can be seen by all but a private one can only be seen by the role categories above. The attributes of a guest can be seen by all but the private attribute of an admin cannot be seen by anyone.

```
1  metamodel: "http://miso.es/dsls/tourismDSL.ecore"
2
3    Admin {
4      image: "http://www.miso.es/images/miso.png"
5    }
6
7    Guest{}
8
9    tourist {
10     // attributes
11     + string name
12     - integer age
13     // permissions
14     @DSL canRead (read) {
15       -> tourism.Cultural
16       -> tourism.Transportation
17       //...
18     }
19     @DSL giveOpinion (write) {
20       -> tourism.Opinion
21     }
22     @DSL changeOpinion (modify) {
23       -> tourism.Opinion.rate
24       -> tourism.Opinion.opinion
25     }
26     @Editor canAddPictures (AddPictures)
27              when closer 10 m
28   }
29
30   touristicguide extends tourist {
31     // attributes
32     + string[*] spokenLanguages
33     // permissions
34     @DSL touristicGuideCanDoAll(all) {
35       -> tourism.Museum
36       //...
37     }
38     //...
39   }
```

**Listing 1: Example of the User Role MM concrete syntax.**

Finally, *permissions can have contextual constraints that will limit them to a certain situation* (R4). For instance, in Figure 5, the class *GeoConstraint* can be used to specify that a tourist can only see (read) an element when it is close to it at less than 5 kilometres or be able to attach pictures to a geopositioned element when it is at less than 10 metres from it. This functionality is proper to mobile devices and, as far as we know, does not exist in traditional role-based access solutions. The expansion of our role model with more contextual constraints will be the topic of future work.

## 4.2 Concrete syntax

Listing 1 shows an example of the concrete syntax that we have implemented for the user role meta-model. It corresponds to the running example introduced in Section 2, and therefore, it annotates the domain meta-model of the tourism DSL. The first line of the listing specifies the URI of this meta-model. This provides access to the meta-model elements to be annotated, and enables auto-completion when using the syntax.

The listing defines four roles: Admin and Guest, which are mandatory, and the custom roles *tourist* and *touristicguide*. The Guest role has not been assigned any permission (line 7), and therefore the access to the models built with the tourism DSL is private, only enabled to users having at least the role of *tourist*. The *tourist* role defines two attributes in lines 11–12. Moreover, it is granted permission to *read* different elements of the domain meta-model, but it can only *write* and *modify* opinions and rates (lines 14–25). Tourist also has permission regarding the editor functionality (line 26) with a contextual constraint which allows a tourist to add pictures to an element when it is closer than 10m to it. The *touristicguide* role inherits from the *tourist* role (line 30). This way, *permissions are inherited between roles* (R1). In addition, *touristicguide* defines additional permissions (lines 34–37).

# 5 TOOL SUPPORT

In this section, we describe the architecture and tool support for the handling of roles. First, Section 5.2 explains the overall architecture, then, Section 5.2 describes the management of roles on the mobile app, and finally Section 5.3 reports on the functionality of the website depending on the roles of users.

## 5.1 Architecture

Figure 6 shows the architecture of the envisioned environment, where some parts of it are still under development. This architecture extends the one described in Section 3.2. The mongoDB database stores the username and password of users. A user can access both the website of DSL-comet and the mobile app with the same username and password; however, the user roles may be different in each case. Each domain DSL has its user role hierarchy stored in a specific database table and they reference the domain DSL with its URI. These user role hierarchies have a set of roles and these roles have a set of reference towards specific users. This way we can know the role of the users depending on the context of the domain meta-model.

We are currently extending DSL-comet to support role-based access control policies defined with the user role meta-model introduced in the previous section. Currently, user role models are defined using an Xtext [5] editor within Eclipse, and then, as we will explain in Section 5.3, they are uploaded to the mongoDB database of DSL-comet via its website. However, our short-term goal is developing a low-code environment featuring a web-based text editor where roles can be more easily defined. We plan to do so based on the existing integration between Xtext and JavaScript libraries like Orion, Ace or CodeMirror[1], or using EMF.cloud [28].

## 5.2 DSL-comet app

Users can use the mobile client of DSL-comet with a registered account or as guests. The latter only grants access to the languages that lack a user role model, or which have public access for guests.

DSL-comet displays the list of Active DSLs available in the server. If a language has specified a user role model, DSL-comet will apply the defined roles and permissions. Given a language, users will be able to see the role hierarchy corresponding to the language, the specific roles assigned to them, and can also *request a specific role to the language author* (R5). For the latter, they need to be logged in. Users who have not been assigned any of the roles defined for a language will be considered guests. Figure 7 shows the pop-up that is displayed when a user clicks on the "+" button for a specific palette (these buttons are observable in the background). The pop-up says

[1] https://www.eclipse.org/Xtext/documentation/330_web_support.html

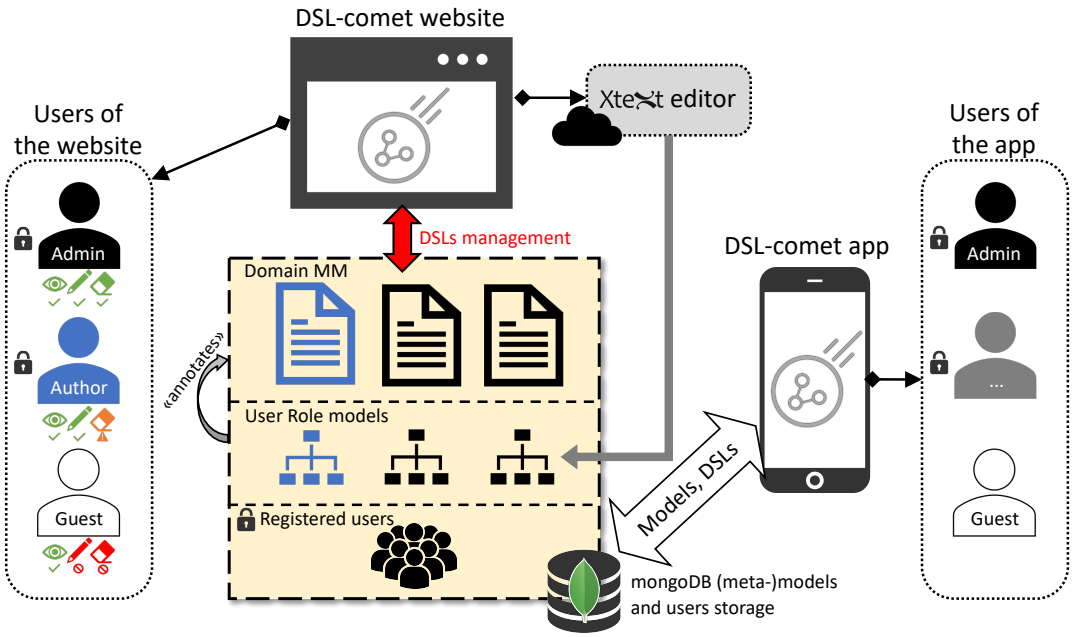



**Figure 6: Management of roles within DSL-comet.**

the role of the user "guest" is Guest for the language TourismDSL and it displays the hierarchy for this language.

Upon selecting a language and logging in with a specific role, DSL-comet will display the language modelling editor. The user will only be able to see and interact with the elements for which the selected user role has the necessary permissions. For example, if the user role lacks the necessary permissions, some buttons in Figure 4 may be disabled, some model elements may be hidden (label 2 of Figure 4), and some elements of the palette may be hidden or not clickable (label 1 of Figure 4).

In geo-located DSLs, users can be part of the model and appear at their current location on a map like in Figure 8. Roles may define attributes (see Listing 1), which users having those roles can fill in. In a collaborative session, users can see each other and consult each other's attribute values depending on their visibility (private or public).

Regarding the example of Section 2, let's describe the scenario applied to DSL-comet. Tourists have to register to DSL-comet and request the role of "Tourist" to the administrator through the app. Once accepted, they can load route models created by the touristic guide in order to see the points of interest and routes geolocated on a map. They can also join a collaboration session initiated by the touristic guide while making the tour. In both cases, tourists cannot move or add elements to the route, but they can use some extended modelling options such as adding pictures attached to elements.

## 5.3 DSL-comet website

The website of DSL-comet is the interface to manage the languages that are to be available in the mobile app. The functionalities that the web application offers depend on the users' identity. The website has three categories of users: guest, author and administrator.

Guests are users that are not logged in. They can only see the language definitions (domain meta-models, graphical syntax models), which are stored on the database. They can also see the instances of these languages that have been saved through DSL-comet app if these languages do not have a user role hierarchy associated or a complete "read" guest role, which implies it can see all elements of a meta-model.

Authors are authenticated users. They can upload new domain meta-models to the database, of which they become their authors. Similarly to guests, they can see the list of all meta-models stored on the database. In addition, they can see the list of their meta-models, delete them and manage how other users access to them. They are also responsible to provide the user role and graphical syntax models annotating their domain meta-models. Currently, the user role models are defined using an Xtext editor, and their internal XMI representation is uploaded to DSL-comet using the website; however, in the future, we plan to create a low-code environment to define these models on the cloud in a user-friendlier way. In addition, *authors also handle the user role requests made by users through the mobile app* (R5), and so, they administrate which users can use their languages on the mobile app. Figure 10 illustrates the latter statement regarding the management of roles, we can see that the user Lissette is the author because he is part of the Admin and there are two touristic guides.

Finally, administrators are authenticated users who have a "superuser" access to all defined meta-models (see Figure 10 which displays the list of existing DSLs, an author would have a reduced list). By default, they are also administrators of every defined language.


**Figure 7: Screenshot of DSL-comet when role hierarchy is displayed**


**Figure 8: User's attributes in DSL-comet.**

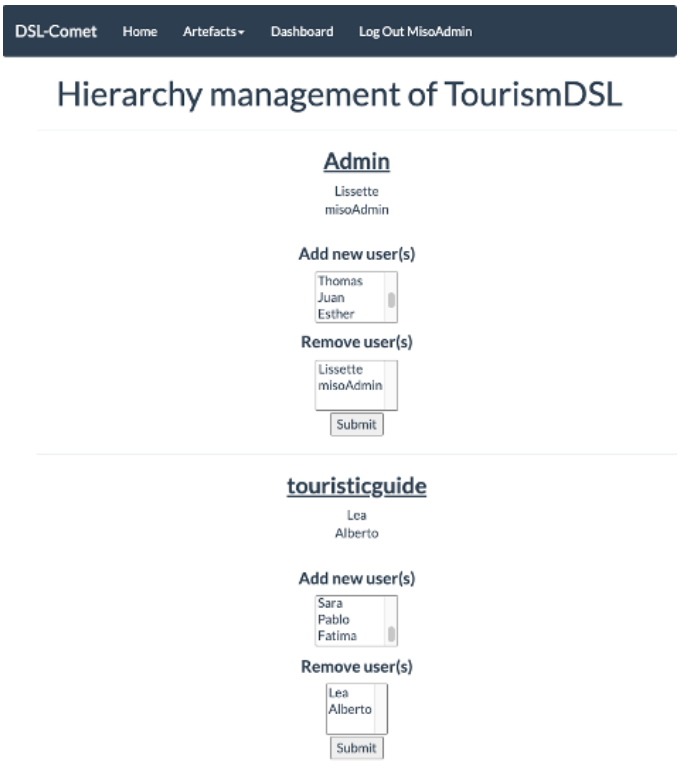



**Figure 9: Screenshot of the management of users and roles for the Tourism DSL**

## 6 RELATED WORK

There are several tools that support collaborative modelling [19] on mobile devices. For example, NetSketcher [1] is a tool to create BPMN models on mobile devices, and it can be used for collaborating in synchronous and asynchronous ways. FlexiSketch [38] is a sketching mobile tool for requirements modelling. Users can draw on a canvas and their sketches are converted into entities that can be typed. Users are allowed to work in collaboration on the same model but different tablets. Socio [18] is another tool for collaborative modelling via social media like Twitter or Telegram. It can be used on mobile devices, and different stakeholders can create a model using natural language using the same thread of conversation. However, none of these tools provides mechanisms to control how users collaborate (i.e., every user has the same permissions). In the following, we revise a bunch of works that do provide some kind of user access control, albeit none of them considers mobility aspects.

Bergman et al. [4] propose a rule-based fine-grained access control approach which is close to our proposal, as they allow establishing rules for checking whether a user is permitted to access some data. They also deal with conflict resolution. However, they do not consider control access for features specific to modelling on mobile devices, such as geoposition or editor-specific actions such as taking pictures for extended modelling.


**Figure 10: Screenshot of the administrator's dashboard on the website**

CDO [27] is an online model repository for EMF models and meta-models. It provides multi-user access and collaboration, and authorization policies can be established programmatically for groups of users or roles. This tool provides role-based access control like us, although it is not managed through a meta-model. Similar to the previous approach, it is designed for static environments.

AToMPM [25] is a framework to generate domain-specific modelling web-based tools that run on the cloud. The tool provides two ways of online collaboration: by sharing the same model and canvas or by sharing the abstract syntax only. The visibility of the collaboration is controlled using user access control at different levels of granularity: package, model and model element.

WebGME [36] proposes to create DSLs directly inside the browser using a UML class diagram-based meta-model. The solution is cloud-based and provides model versioning and online collaboration. Shared data are protected by using user access control.

Our work is based on the RBAC model [11], extended with role attributes, and geo-positioning constrains stating when certain permission is granted to a user on a given object. While this is not to be considered a full-fledged contextual modelling language for model adaptation (since this would correspond to the context meta-model in Fig. 2), some works can be found on modelling context adaptation for mobile or web applications [8, 9, 21, 26]. We will take inspiration for those works to develop our DSL to specify contextual rules for model adaptation.

Altogether, on the one hand, we have found applications to create models collaboratively on mobile devices, and on the other hand, we have found solutions to provide role-based access control for desktop modelling applications. However, to the best of our knowledge, there are no proposals to permit safe collaborative modelling on mobile devices. The fact that users can be geo-located in models, and that roles can define attributes for users, is another distinguishing feature with regards to access control approaches on desktop environments.

## 7 CONCLUSION

Modelling is an activity which can be done collaboratively. This can raise some issues when some stakeholders are not supposed to see all the elements of a model or if some users should not alter some part of the model.

In this paper, we have described an approach to provide role-based access control to collaborative modelling on mobile devices. We have proposed a user role meta-model that annotates a domain meta-model and extends the notion of Active DSL with role hierarchies and their permissions. We have also presented the first steps towards its implementation within the mobile modelling tool DSL-comet.

We are currently finishing the implementation of our proposal for managing roles. In the future, we plan to develop a low-code development environment to facilitate the definition of Active DSLs on the cloud, so that citizen-users can easily define and deploy them on DSL-comet. In addition, we aim to fully define the meta-models for defining contextual rules able to adapt the model at run-time.

## ACKNOWLEDGMENTS

This work has been funded by the the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement n° 813884 (Lowcomote[30]), by the Spanish Ministry of Science (project MASSIVE, RTI2018-095255-B-I00) and the R&D programme of Madrid (project FORTE, P2018/TCS-4314).

## REFERENCES

[1] Nelson Baloian, Gustavo Zurita, Flávia Maria Santoro, Renata Mendes de Araujo, S. Wolfgan, D. Machado, and José A. Pino. 2011. A collaborative mobile approach for business process elicitation. In *CSCWD*. IEEE, 473–480.
[2] Barbara Rita Barricelli, Fabio Cassano, Daniela Fogli, and Antonio Piccinno. 2019. End-user development, end-user programming and end-user software engineering: A systematic mapping study. *J. Syst. Softw.* 149 (2019), 101–137.
[3] Gábor Bergmann, Csaba Debreceni, István Ráth, and Dániel Varró. 2016. Query-based access control for secure collaborative modeling using bidirectional transformations*. In *MoDELS*. ACM, 351âĂŞ361.
[4] Gábor Bergmann, Csaba Debreceni, István Ráth, and Dániel Varró. 2017. Towards efficient evaluation of rule-based permissions for fine-grained access control in collaborative modeling. In *COMMitMDE@MoDELS (CEUR Workshop Proceedings, Vol. 2019)*. CEUR-WS.org, 135–144.
[5] Lorenzo Bettini. 2016. *Implementing domain-specific languages with Xtext and Xtend.* Packt Publishing Ltd.
[6] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-Driven Software Engineering in Practice, Second Edition.* Morgan & Claypool Publishers.
[7] Frank Budinsky, Stephen A. Brodsky, and Ed Merks. 2003. *Eclipse Modeling Framework.* Pearson Education.
[8] Stefano Ceri, Florian Daniel, Maristella Matera, and Federico Michele Facca. 2007. Model-driven development of context-aware Web applications. *ACM Trans. Internet Techn.* 7, 1 (2007), 2.
[9] Sylvain Degrandsart, Serge Demeyer, Jan Van den Bergh, and Tom Mens. 2014. A transformation-based approach to context-aware modelling. *Software and Systems Modeling* 13, 1 (2014), 191–208.

[10] Sebastian Döweling, Tarik Tahiri, Benedikt Schmidt, Alexander Nolte, and Mohammadreza Khalilbeigi. 2013. Collaborative business process modeling on interactive tabletops. In *ECIS*. 29.

[11] David F. Ferraiolo and D. Richard Kuhn. 1992. Role-Based Access Controls. In *NCSC*. 554–563.

[12] Mirco Franzago, Davide Di Ruscio, Ivano Malavolta, and Henry Muccini. 2018. Collaborative model-driven software engineering: A classification framework and a research map. *IEEE Trans. Software Eng.* 44, 12 (2018), 1146–1175.

[13] Jesús Gallardo, Crescencio Bravo, and Miguel A. Redondo. 2012. A model-driven development method for collaborative modeling tools. *J. Netw. Comput. Appl.* 35, 3 (2012), 1086–1105.

[14] Steven Kelly and Juha-Pekka Tolvanen. 2008. *Domain-specific modeling - Enabling full code generation.* Wiley.

[15] A. J. Ko, Robin Abraham, Laura Beckwith, Alan F. Blackwell, Margaret M. Burnett, Martin Erwig, Christopher Scaffidi, Joseph Lawrance, Henry Lieberman, Brad A. Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The state of the art in end-user software engineering. *ACM Comput. Surv.* 43, 3 (2011), 21:1–21:44.

[16] Remo Lemma, Michele Lanza, and Andrea Mocci. 2015. CEL: Touching software modeling in essence. In *SANER*. IEEE Computer Society, 439–448.

[17] Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and how to develop domain-specific languages. *ACM Comput. Surv.* 37, 4 (2005), 316–344.

[18] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2018. Collaborative modeling and group decision making using chatbots in social networks. *IEEE Software* 35, 6 (2018), 48–54.

[19] Michiel Renger, Gwendolyn L. Kolfschoten, and Gert-Jan de Vreede. 2008. Challenges in collaborative modelling: a literature review and research agenda. *Int. J. Simul. Process. Model.* 4, 3/4 (2008), 248–263.

[20] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. 1996. Role-based access control models. *IEEE Computer* 29, 2 (1996), 38–47.

[21] Sigrid Schefer-Wenzl and Mark Strembeck. 2013. Modelling context-aware RBAC models for mobile business processes. *Int. J. Wirel. Mob. Comput.* 6, 5 (2013), 448–462. https://doi.org/10.1504/IJWMC.2013.057387

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

[23] Alberto Sebastián-Lombraña, Esther Guerra, and Juan de Lara. 2020. Positioning-based domain-specific modelling through mobile devices. In *SEAA*. to appear.

[24] Ronny Seiger, Maria Gohlke, and Uwe Aßmann. 2019. Augmented reality-based process modelling for the Internet of Things with HoloFlows. In *EMMSAD (LNBIP, Vol. 352)*. Springer, 115–129.

[25] Eugene Syriani, Hans Vangheluwe, Raphael Mannadiar, Conner Hansen, Simon Van Mierlo, and Hüseyin Ergin. 2013. AToMPM: A web-based modeling environment. In *Demos/Posters/StudentResearch@MoDELS (CEUR Workshop Proceedings, Vol. 1115)*. CEUR-WS.org, 21–25.

[26] Chantal Taconet and Zakia Kazi-Aoul. 2010. Building Context-Awareness Models for Mobile Applications. *J. Digit. Inf. Manag.* 8, 2 (2010), 78–87.

[27] The Eclipse Foundation. CDO. 2020. https://www.eclipse.org/cdo/.

[28] The Eclipse Foundation. EMF.cloud. 2020. https://www.eclipse.org/emfcloud/.

[29] The Eclipse Foundation. Sirius. 2020. https://eclipse.org/sirius.

[30] Massimo Tisi, Jean-Marie Mottu, Dimitrios S. Kolovos, Juan de Lara, Esther Guerra, Davide Di Ruscio, Alfonso Pierantonio, and Manuel Wimmer. 2019. Low-comote: Training the next generation of experts in scalable low-code engineering platforms. In *STAF (Co-Located Events) (CEUR Workshop Proceedings, Vol. 2405)*. CEUR-WS.org, 73–78.

[31] Ash Turner. 2020. How many smartphones are in the world? https://www.bankmycell.com/blog/how-many-phones-are-in-the-world.

[32] Diego Vaquero-Melchor, Antonio Garmendia, Esther Guerra, and Juan de Lara. 2016. Domain-specific modelling using mobile devices. In *ICSOFT, Revised Selected Papers (CCIS, Vol. 743)*. Springer, 221–238.

[33] Diego Vaquero-Melchor, Antonio Garmendia, Esther Guerra, and Juan de Lara. 2016. Towards enabling mobile domain-specific modelling. In *ICSOFT*. SciTePress, 117–122.

[34] Diego Vaquero-Melchor, Javier Palomares, Esther Guerra, and Juan de Lara. 2017. Active domain-specific languages: Making every mobile user a modeller. In *MoDELS*. IEEE Computer Society, 75–82.

[35] Markus Voelter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart C. L. Kats, Eelco Visser, and Guido Wachsmuth. 2013. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org. http://www.dslbook.org

[36] WebGME. 2020. https://webgme.org.

[37] Y Wu. 2016. Global smartphone user penetration forecast by 88 countries: 2007-2022. *Wireless Smartphone Strategies Services* (2016).

[38] Dustin Wüest, Norbert Seyff, and Martin Glinz. 2019. FlexiSketch: a lightweight sketching and metamodeling approach for end-users. *Software and Systems Modeling* 18, 2 (2019), 1513–1541.