Introducing the MOSKitt User Interface Modeling (UIM) Language

Joan Fons[a], Begoña Bonet[b], Marc Gil[c], Javier Muñoz[c], Vicente Pelechano[a]

[a]{jjfons,pele}@dsic.upv.es. ProS Research Center. Universidad Politécnica de Valencia
[b]bonet_beg@gva.es. Conselleria de Infraestructuras y Transporte. Generalitat Valenciana
[c]{jmunoz,mgil}@prodevelop.es, Prodevelop S.L.

INTRODUCTION

The goal of the MOSKitt User Interface Modeling Language (MOSKitt-UIM) is to support the abstract specification of user interfaces for information systems without taking into account neither specific platform details nor graphical design issues. MOSKitt-UIM has been defined in the context of the MOSKitt project[1]. The MOSKitt project aims to provide a set of open source tools and a technological platform for supporting the execution of software development methods which are based in model driven approaches, including graphical modeling tools, model transformations, code generation and collaboration support. This project is led by the Conselleria de Infraestructuras y Transporte (CIT) (Ministry of Infrastructures and Transport) of the Regional Government of Valencia, in Spain. The project was initiated in 2007 and their results are currently in practice by CIT and other companies and ministries in Spain. MOSKitt is participating in the Eclipse Papyrus project, which aims to develop a completely usable UML2 graphical editor in the context of the Eclipse project, and also in national and European research projects, like ATENEA, ITEA2 OSAMI[2] and ITEA2 UsiXML[3].

The definition of MOSKitt-UIM is led by the ProS Research Center[4] of the Universidad Politécnica de Valencia. ProS Center have previously defined other model driven software languages, like the Object Oriented Web Solutions (OOWS) for modeling web sites. They were involved in the Web Engineering Network of Excellence (WEENet) and have experience transferring knowledge to industrial tools. CIT is specifying the language requirements and providing input from a user point of view. They are applying model based approaches for many years, including user interface modeling using stereotyped UML models. Prodevelop implements the tool that supports the language and also provides technological feedback. Prodevelop has a solid background in the development of complex information systems, with a special focus in systems for organizations that manage docking facilities. In summary, MOSKitt-UIM is applying in an industrial production environment the current scientific approaches that promote the definition of user interfaces using abstract concepts.

Although the final goal of MOSKitt is to automatically generate functional applications, MOSKitt-UIM also aims to support the definition of specifications that can be taken as input by programmers to manually implement the application user interface.

USER INTERFACE MODELING APPROACH

MOSKitt-UIM is focused on the definition of complex user interfaces for information systems. Therefore, the language assumes that a data model and a functionality model have been specified somehow (UML2, DB, BPMN, etc.). These models must provide information about information entities (with properties and relationships between

[1] http://www.moskitt.org/eng/
[2] http://www.osami-commons.org/
[3] http://itea.defimedia.be/
[4] http://www.pros.upv.es/

them) and operations. Using MOSKitt-UIM an analyst defines how this data and this functionality is accessible to system users and other user interface behavior.

A MOSKitt-UIM specification is organized in Views, which are parts of the overall user interface. System analysts specify which Views can be accessed for each kind of user (administrator, anonymous, IT department user, etc.) defining the visibility that each user will have. A system View contains a set of Interaction Units (IUs). An IU describes a specific interaction between the system and this user. MOSKitt-UIM defines several Interaction Units types, depending of the kind of interaction that these IUs support:

- *Information IU*: in an Information IU the system shows data to users (in a table, in a registry or using any other kind of presentation).
- *Operation IU*: in an Operation IU the system gathers an operation parameters and invokes that operation.
- *Navigation IU*: in an Navigation IU the user can use navigation mechanisms (links) to get other IUs.
- *Selection IU*: in a Selection IU users can select one (or many) entities from a entities population.
- *Editable Information IU*: in an Editable Information IU the system shows information to users, and some of that information can be used as parameters to invoke an operation. Therefore, it is an Information IU and an Operation IU at the same time.

Additionally, several IUs can be grouped in a Composed IU to represent complex IUs (like, for instance, Master/Detail screens). All these concepts are described with more detail in the following sections.

In some organizations which must deal with many software development projects, like CIT, it is a common practice to define rules or **patterns** for implementing user interfaces. Applying this approach, system analysts do not define every Interaction Unit from scratch, but they have to follow the organization guidelines in order to describe the user interface. In order to support this practice, MOSKitt-UIM provides mechanisms to allow system analysts to define and apply their own user interface patterns. Pattern Definitions are used by the method manager in the organization to specify pre-configured Interaction Units (which can be complex IUs). For instance, a Pattern Definition can be specified to represent a Master/Detail screen, where an Information IU plays the role of Master and another Information IU plays the role of Detail. These Pattern Definitions are applied by system analysts using Pattern IUs. The Patterns IUs are completed by the analysts to represent the specific user interface that they are describing. For instance, the system analyst will specify which is the information shown in the Master Information IU and which is the information shown in the Detail Information IU.

Patterns play a key role when MOSKitt-UIM is applied in the context of a model driven method for software development. Patterns can be used to tailor the language building blocks to an specific meaning in an organization and, therefore, they provide valuable knowledge to code generators (or model transformations) which are able to produce output that fits the needs of that organization. Anyway, MOSKitt-UIM has been designed having in mind also an scenario where the specification produced by an analyst is taken as input by programmers who will manually code the final user interface.

MOSKITT-UIM OVERVIEW

As it has been briefly introduced in the previous section, Interaction Units are the main building blocks when describing an application user interface. Interaction Units can be intuitively mapped into application windows or web pages, but they represent an abstract concept, so they could be finally implemented as several windows attending to, for instance, specific technology or device constraints. In this section we introduce the main concepts of every Interaction Unit in order to provide an overview of the language.

Information Interaction Unit

An Information IU is used to specify an interaction where the system shows information to users. The information that must be shown is described in Visualization Sets. A Visualization Set describes a view of the system data entities. Therefore the Visualization Set includes Class Views[5] that represents the business entities.

[5] In fact, those Class Views are a view over classes in a UML Class Diagram, Entities in an ER/Database model or Entities in an Ecore Model.

Class Views include Visible Attributes that specify which are the properties of an entity that must be shown in a Visualization Set. A Visualization Set could show information from different data entities. In that case several Class Views must be included. These Class Views must be related using Recovery Relationships, defining the dependency that exists between those classes. To describe the functionality that can be invoked in that Information IU, Visible Operations that are linked to operations in the business layer can also be specified.

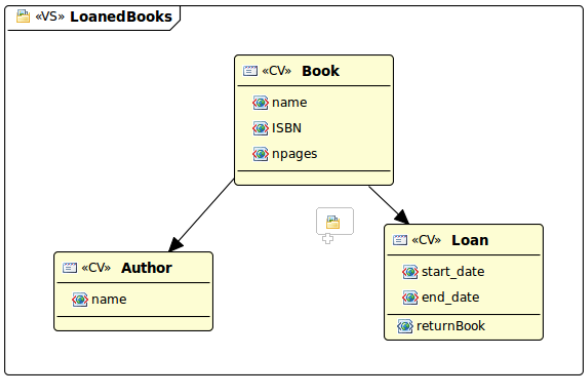

Figure 1. A Visualization Set in an Information IU that shows Books and Loans.

Figure 1 shows a Visualization Set which specifies the contents of an Information IU. This Visualization Set defines an Interaction Unit that shows *Book* information (its *name*, *ISBN*, and *number of pages*), including the name of its related *Author*, and their current Loan information (start_date and end_date). A Visible Operation *returnBook* has been specified in the *Loan* Class View to represent that books can be marked as returned from this Interaction Unit.

Additional information of a Visualization Set can also be specified in MOSKitt-UIM, like ordering and filtering criteria, pagination guidelines (for instance, to show five loaned books in each page), masks for visualizing Property Views and Navigations to navigate to other Interaction Unit when selecting and entity (for instance, navigating to a screen that shows more book information where clicking in the name of a book). User interface behavior is specified using ECA rules, where the interface respond to events in its components to perform actions if a condition holds. Those events and actions have been defined as extensible elements, so that any organization is able to define their own elements.

Operation Interaction Unit

An Operation IU is used to gather an Operation Parameters (a service in the business layer) and to invoke that operation. Operation Parameters can be initialized when the value should not be introduced by users. When an Operation Parameter is an entity, a Selection IU can be related to the parameter to describe how the entity must be selected. Additionally, analysts can define constraints and validations for a parameter, and visualization and editions masks.

Editable Information Interaction Unit

An Editable Information IU is a combination of an Information IU and an Operation IU. In an Editable Information IU information is shown to users, like in an Information IU, but some of that information can be used as input parameters for invoking system services, like in an Operation IU. Therefore, the specification of an Editable Information Unit involves the specification of a Data Binding that maps the Visible Attributes defined in a Visualization Set with the Operation Parameters of an Operation.

Navigation Interaction Unit

A Navigation Information IU are used to define navigations between Interaction Units. Usually, a Navigation Information IU will be implemented as a menu. They can represent complex navigation structures with several levels. Navigation Information IUs may be used for specifying the root system menu or as a menu block in composed interaction units.

Selection Interaction Unit

Analysts use Selection IUs to describe how one (or many) entities must be selected from an entity population. This kind of IU is used in conjunction with Operation or Editable Information IUs to support the selection of entities in Operation Parameters and Filter Parameters. They can be seen as an specialization of an Information IU with the additional behavior of allowing to select elements. Therefore, it includes the specification of a Visualization Set that describes the data to be shown.

Pattern Interaction Unit

As it has been argued in previous sections, organizations or, in general, development teams, tend to define guidelines for describing their user interfaces. In order to support this practice with MOSKitt-UIM, the language introduces the concept of Pattern. Method managers can define Pattern Interaction Units that analysts will be able to use when describing their user interfaces. When defining a Pattern, several pre-configured Interaction Units can be included by specifying, if necessary, the role that they play in the pattern. For instance an Information IU can play the *master role* in a Master/Detail pattern whereas other Information IU can play the *detail role*. Additionally, multiplicity constraints can be defined to express the minimum and maximum times that a role or kind of IU can or must appear when applying the pattern.

TOOL SUPPORT

MOSKitt-UIM supports the specification of complex user interfaces and, therefore, it is a complex language with more than 80 concepts. Such a complex language must be supported by a powerful tool in order to be applied by users in real scenarios. MOSKitt-UIM is supported by a graphical tool that is integrated in the MOSKitt environment. Since MOSKitt is based on Eclipse, the MOSKitt-UIM editor has been implemented as a set of Eclipse plugins that use or extend Eclipse technologies. The MOSKitt-UIM metamodel has been implemented using Ecore, the metamodeling language provided by the Eclipse Modeling Framework (EMF). Using the capabilities that are provided out-of-the-box by EMF, MOSKitt-UIM models are stored using an XML format.

The MOSKitt graphical editor, which is shown in Figure 2, has been implemented using the Eclipse Graphical Modeling Framework (GMF). The graphical editor that is automatically generated by GMF has been extensively extended to increase the functionality and usability. Additionally, a tree-like model browser has been implemented and powerful property sheets are available to edit the elements properties.

As it has been described in previous sections, MOSKitt-UIM relies on an existing data and functionality model. The MOSKitt-UIM tool supports the definition of user interface models that are based on UML2, database, ECore or BPMN models. Since others base models could be of interest in other organizations, the tool defines an Eclipse extension point to easily support other kind of information or functionality models.

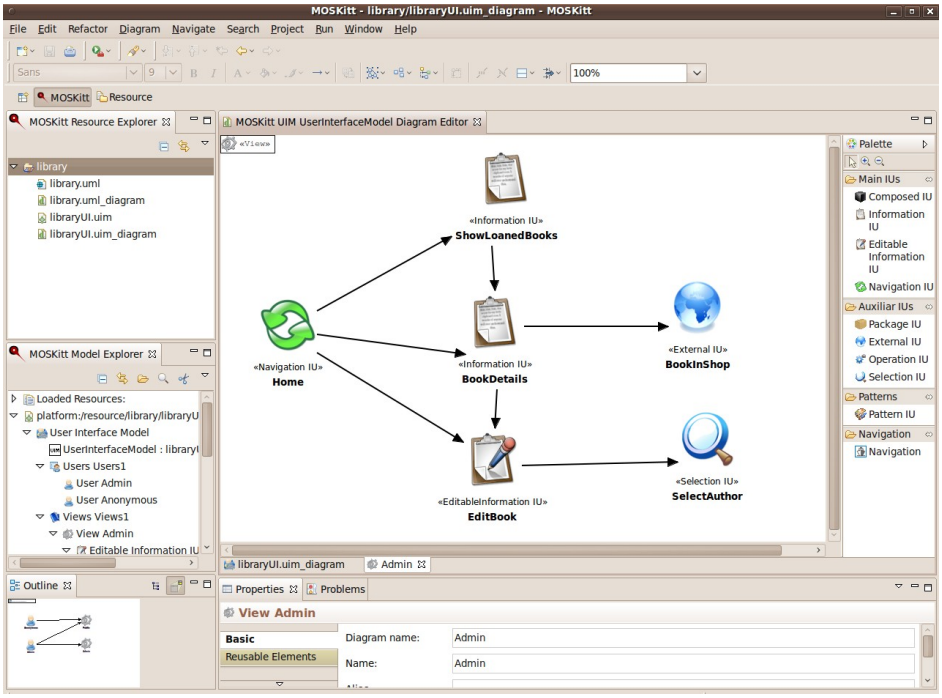

Figure 2. The MOSKitt-UIM graphical editing tool

Additionally, the prepackaged MOSKitt environment includes and adaptation plugin which extends MOSKitt to support the application of gvMetrica, the CIT method for developing information systems. In this adaptation, MOSKitt-UIM patterns which follow the CIT guidelines can be found. These patterns can be applied out-of-the-box to model user interfaces or can also be used as examples to define other specific patterns. Taking as input the CIT patterns, code generation capabilities are currently available which produce PHP code from UIM models.

CONCLUSIONS AND FUTURE DEVELOPMENTS

MOSKitt-UIM is a young modeling language for user interfaces which has been created with the advisement of the academia and taking as input industrial requirements. The language is currently in practice in CIT and the development team is continuously receiving feedback for improving the usability of the language and the tools.

In order to provide a more complete user interface modeling solution, the MOSKitt team is currently designing an extension to support wireframe prototyping. These prototypes will be used by analysts to show to the users a potential presentation of Interaction Units. Analysts will also be capable of adding additional information (like selecting specific kinds of widgets for representing a property) which will be able to guide the (automatic or manual) implementation process.