MDA based Hypermedia Modeling Methodology using reusable components

Pablo Vera[1], Claudia Pons[2], Daniel Giulianelli[1], Rocío Rodríguez[1]

[1] National University of La Matanza
Department of Engeniering and Technological Research
San Justo, Buenos Aires, Argentina
{pvera, rrodriguez, dgiulianelli}@ing.unlam.edu.ar
[2] National University of La Plata
LIFIA – Research and Education Laboratory on Advance Computing
La Plata, Buenos Aires, Argentina
cpons@lifia.info.unlp.edu.ar

Abstract. This paper shows a modeling and developing methodology for mobile web applications using the MDA (Model Driver Architecture) approach. It´s based on a conservative extension of UML allowing modeling with any existing tool capable of exporting the model to XMI (XML Metadata Interchange), the standard exchange format for UML diagrams. The methodology is based on the four main activities of the hypermedia modeling: conceptual modeling, navigation modeling, abstract interface design and implementation, but it simplifies the interface and navigation modeling based on reusable components. These components will be easily automatically converted into the application source code which with the use of different templates can target different platforms and also have a front-end and back-end system on different platforms for the same application.
Keywords: MDA, Mobile Web Applications, Mobile, UML

1 Introduction

1.1 Hypermedia Modeling

Modeling of hypermedia systems is a practice that started several years ago when the web was starting to be massively adopted. One of the first works on this area is OOHDM (Object Oriented Hypermedia Design Method) [8] which established four main activities on the hypermedia design. These activities are: conceptual modeling, navigation modeling, abstract interface design and implementation.

Conceptual design established the different classes that will compose the application´s domain model. The most used technique is UML diagram classes like in OOHDM and UWE (UML Based Web Engineering) [5]. Other works uses entity relation diagrams like in WebML (Web Modeling Language) [1] and RMM (Relationship Management Methodology) [3].

Navigation design consists of building a model that shows the different path that the user can take when using the application. This point is proper to hypermedia

systems where the user has a non linear system where he or she can follow different links for browsing system entities, like for example, hypertext in a website. Most of existing methodologies uses extended UML classes' diagrams that include stereotypes to mark navigation classes. So navigation is derived from structural models. MDHDM (Model Driven Hypermedia Design Method) [7] complements navigation using process models that enable business logic to guide user navigation making it more dynamic.

Next activity is abstract interface design that defines, the way that the user will see and will interact with the application. OOHDM uses ADV (Abstract Data Views) [2] where different objects are grouped to create a screen mockup. WebML defines its own graphic notation to define the mockups. This notation is based on XML which makes the process of transforming the mockups into usable code easier, for example HTML.

Finally, implementation is about the development of the application based on the models. At this point the different methodologies attempt to automate as much as possible the resulting code. WebML do it using XSLT transformations (Extensible Stylesheet Language Transformation) [11]. OOHDM establish the required mapping that must be made from each of the models to generate code and proposes a tool that allows designing an application using the model for generating the application code. MDHDM performs transformations to create Java source code.

1.2 Mobile Web Applications

The most widespread hypermedia systems are websites where the user can visualize: images, text, multimedia components besides browsing the pages through hypertext.

A web application is a webpage including some type of server side scripting. A particular case of web applications are those specially design for mobile devices and the most common of mobile devices are cell phones. Despite the great evolution of cell phones that have high processing power, they still have the following limitations and needs: (1) Small screen; (2) The need of simple controls; (3) Simply and directly display information, without the complex screen layouts of conventional web sites; (4) Reduced, practical and intuitive navigation system; (5) Simple text input.

W3C consortium (World Wide Web Consortium) has a guide of best practices of mobile web sites [10] and mobile web applications design [9] which provides the best way to settle the points mentioned above to achieve a usable website from any mobile device.

2 Proposed Modeling Methodology

Based on the premise of designing a suitable web application for mobile devices, a new hypermedia modeling methodology is proposed. The methodology simplifies the interface and navigation design based on reusable components. These components will be easily automatically converted into the application source code. Mobile web applications were chosen because of the screen limitations making it simple to create common components with reduced functionality and avoiding defining complex layouts that cannot be shown in a mobile device or if they can be shown they are not

usable at all. Nevertheless it could be applied to model traditional web applications especially on backend systems where the layouts generated by this tool will be enough for an internal system used by some administrators.

The methodology is based on the activities proposed by OOHDM but unifying navigation and user interface modeling and it final goal is to generate an application completely automatically from the models. All models are defined in UML, creating a conservative extension of the language. Figure 1 shows the stages of the methodology which is based on MDA approach by creating diagrams and transforming them with an automatic tool through different stages. Models created by the user are marked in the figure with a person icon.

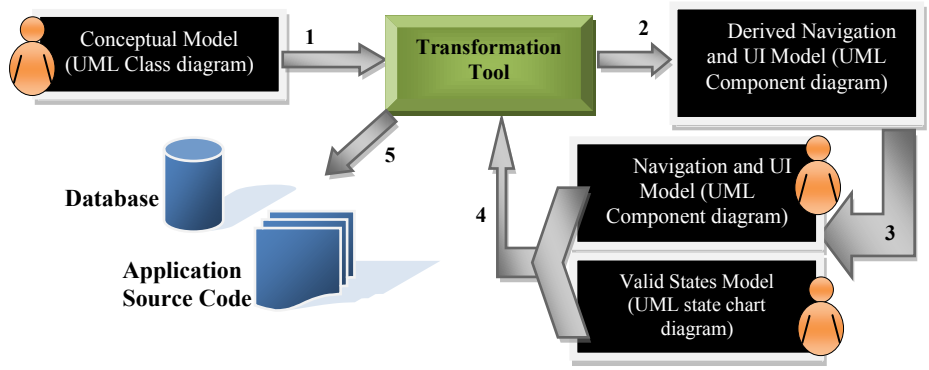

Fig. 1. Methodology Stages

The following sections explain each stage of the proposed methodology.

2.1 Conceptual Modeling

Like in methodologies previously mentioned the conceptual design will be made using an UML class diagram but extending it to make the database generation from it easier.

(A) **Special properties:** From class properties two main attributes will be remarked: the unique identifier of the object and the descriptor. The identifier corresponds with a unique key that identifies uniquely the class instance or the database register when using a relational data model. The identifier will be also used to map the relations between the different tables of the database. The descriptor will be the property that contains a user readable description of the object, for example a person´s name, a book´s title, etc. A class may not contain a descriptor in some cases but the identifier is always required. To identify those properties, two stereotypes were created: `<identifier>` and `<descriptor>`.

(B) **Enumeration Classes:** In some applications it will be necessary to identify those classes representing limited values, like for example the different states of an invoice, a list of allowed actions, etc. That´s the reason why is necessary the use of classes of enumeration type that must be distinguished from other classes because they will have a special treatment when generating the source code from

them. In order to identify those classes the UML stereotype `<enumeration>` will be used.

(C) **Special Data Types:** In order to improve user experience when using the application, especially from a mobile device, two special data types will be used: `phoneNumber` which is a string representing a telephone number allowing to render a link to automatically make a phone call to that number; `address`: this data type will allow using the GPS device available on advance mobile phones and take the exact location where the user is located. If the GPS is not available or is temporally out of service, it will be possible to enter the address like a string. Also this data type will allow using the GPS to navigate to that location.

2.2 Navigation and User Interface Design

This activity will use UML component diagrams. A set of components is defined. All components are extended with stereotypes and tagged values to allow configuring them to be adapted to the needs of the applications being modeled.

Two common tagged values are present in all the components: `id:` is a value that uniquely identifies a component. It will be useful when referring the component from others for example with designing navigation; `navigation`: it defines the navigation bar that must be present in all components or web pages of the application. It´s important to consider that because when creating a mobile web application, navigation system must be simple and must be present in all web pages. Based on W3C guidelines and best practices the configuration of the navigation is added on each component.

Another tagged value present in most of the components is `MainEntity` which is used to establish the main class of the domain model from which to component will be based. For example an operation of data update will be based on a particular domain entity, whether you can combine data from several classes it always originates from an entity that is the base of the query.

(A) **Components**: The defined components are: `Login`, `List`, `Search`, `Menu`, `CRUD` (Create, Read, Update, Delete), and `UpdateView`. Below is the explanation of each one:

- `Login`: a component to authenticate the user on the system. It´s tagged values are: `MainEntity`, `RedirectToComponent`, `User` and `Password` where the latter two defines the properties of the domain class where the user credentials will be checked. Authenticated user will be accessible from rest of the system components using the special variable called `LOGGEDUSER`. The `RedirectToComponent` value has the id of the component where the user will be redirected after a successful login.
- `List`: Represents a list of information that can be visualized like a grid or a table with rows and columns showing entities information. It´s tagged values are: `MainEntity`, `FixedFilters`, `Columns` and `Sort`.

- FixedFilters defines the data filters to be applied to obtain the list. These filters are applied over the properties of the main entity being able to access related classes using object notation.
- Columns defines each of the columns to be shown in the list, it refers to the properties of the main entity and its related entities, but additionally it´s possible to apply functions to generate calculated data. The columns will also be able to contain links to other components or external pages.
- Sort defines the properties by which the list will be sorted. It´s also possible to specify a calculated column name previously detailed in the columns list.

- **Search**: this component is similar to a List component, except for its capacity of adding filters that instead of being predefined are entered by the user when using the application, allowing performing a query over the data. The result of that query will be a list of information. It´s tagged values are: MainEntity, SearchFilters, FixedFilters, Columns and Sort. Most of the tagged values are identical to the ones defined in the List component. The only added tagged value is SearchFilter that includes search filters entered by the user at runtime. This tagged value specifies the properties on which the query will be performed, been able to define for each one a different type of filter allowing the following values: SingleSelection, MultipleSelection, FreeText, BooleanType, DateRange;
- **Menu**: is a component that defines a menu with links to other components. It contains the tagged value Options that defines the different links that will create the menu options to be shown to the user.
- **CRUD**: this component is responsible of creating, showing, updating or deleting an object of a class. It´s tagged values are: DefaultValuesCreate, MainEntity, SkippedPropertiesCreate, DefaultValuesUpdate, SkippedPropertiesUpdate. If there is certain information that must be completed in the object but it´s not required by the user, DefaultValuesCreate or DefaultValuesUpdate are used, allowing for example to automatically complete the user who created the record and the date and time of modification. SkippedPropertiesCreate or SkippedPropertiesUpdate are those properties that must remain untouched, for example: when updating a record the creating date of the record remains untouched. The behavior of the component will be given by a parameter that will contain the link causing its activation. This link should have the parameter Action, being the possible values of that parameter C, R, U and D, where each of the letters indicates a possible action: C (create), R (read/show), U (update) and D (delete). Aditionally this component must receive the parameter ObjectID for update, read and delete operations that will refer to the primary key of the object where the action will be performed.
- **UpdateView**: represents an update operation with special characteristics. Many times the update of an object is made in parts. For example when an object changes from different states, usually some properties must be filled and others doesn´t. It´s tagged values are: UpdateProperties, DisplayProperties, CreateEntity, DefaultValuesUpdate, SkippedPropertiesUpdate. UpdateProperties are the properties that

will be asked the user to update on this operation. `DisplayProperties` are the properties that will be shown to the user in a readonly mode. `DefaultValuesUpdate` and `SkippedPropertiesUpdate` are similar to the ones of `CRUD` component. `CreateEntity` is an optional tag that is used when the update of the main entity cause the creation of an object of a related class. This is useful for example for log entries.

(B) **Links:** Are defined by a function with parameters of the form: `Link ([link text], [destination], [parameters], [access key])`. An additional function is defined to create optional links called `OptionalLink`.

(C) **Functions:** Are mostly used on List and Search components. They are used to generate columns with data that cannot be directly retrieved from a class attribute, but through a calculation on related classes. Functions always work starting from the main entity of the component searching that entity on the related classes. So the filter relating the main entity with the related class is implicit. Defined functions are: `Sum, Count, Exist, Not Exist, Eval, GetId and Retrive.`

2.3 Modeling Valid States

Usually applications have objects that changes from one state to another and the valid states has a sequence that must be followed. For that cases the UML states diagram will be used applied to a property of a class, showing the valid sequence (usually expressed by the enumeration values of a class). On the Implementation activity business rules will be created to ensure that the sequence is followed, if not an error message will be thrown to the user when trying to set an invalid status to an object.

2.4 Implementation

The implementation activity consists in the building of the application code starting from the models, using Model Driven Architecture (MDA) [4]. All models are based on UML so it´s possible to make the modeling using any existing tool that has the ability to export the model to XMI [6]. XMI is the standard exchange format for UML diagrams. XMI is based on XML so it can be read with any UML Parser.

The starting point is the Domain Model which resulting XMI will be used to perform the first transformation resulting in a partial Component Model including `CRUD` components for each non enumeration classes and a `Menu` linking those components. Then the component diagram will be completed by the user.

Once the component diagram is finished, its XMI will be imported in the transformation tool to generate the application source code and the database script.

The transformation for generating the application source code will be based on a template. A template must be selected to choose the desire resulting source code. This will allow generating code for different platforms using the same models.

So using templates approach will give the opportunity not only to generate code for creating a web application but also a native application can be created programming the appropriate template.

3 Using the methodology

This section shows a brief example of an application modeled with this methodology. The application is a mobile system for taxi drivers where the driver can use the cell phone to see and accept available trips. A driver can accept more than one trip at the same time because of proximity issues but only on trip will be started at the same time.

The application will have a mobile frontend and a standard web backend for administration. The administrator will add new trips to the system and the drivers will access those trips from their mobile devices. Figure 2 shows the classes of the domain model.

Fig. 2**.** Domain Model (UML Class Diagram)

The XMI of the model on Figure 2 will be automatically transformed in a first version of the domain model including `List` and `CRUD` components of all non enumeration classes and also a `Menu` component is generated pointing those components. Figure 3 shows the Navigation and User Interface diagram for the Administration system. Components painted with darker color will be automatically generated, so only three components must be manually created and only minimal changes must be applied to Trip list to create a filtered list of trips called Current Trips.

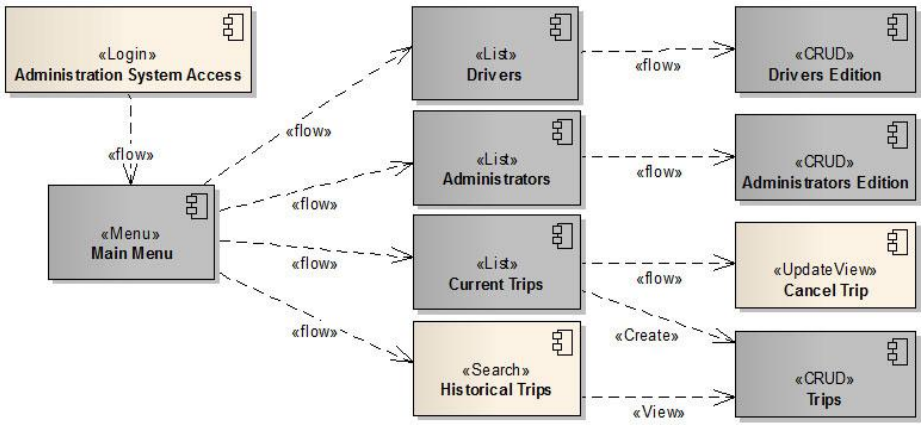

Fig. 3. Navigation and User Interface Model for Administration system (UML Component Diagram)

Modeling the mobile system will require the creation of an additional Component diagram where a few components can be copied from the Administration Model.

Figure 4 shows the resulting model for the mobile system. The components diagram will be imported in the code generating tool allowing selecting different templates for each one to target different platforms.

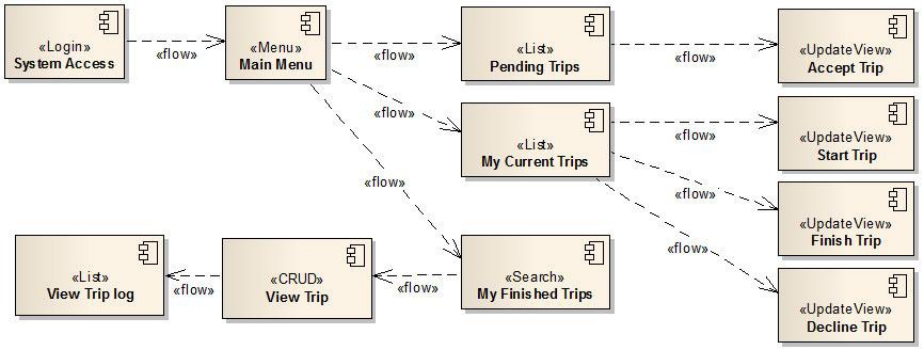

Fig. 4. Navigation and User Interface Model for mobile system

As an example the following tables shows the tagged values required for configuring some of the components of the mobile model.

Table 1: Tagged values for "System Access" component of type Login

Tag	Value
Id	cpnLogin
RedirectToComponent	cpnMainMenu
MainEntity	Driver
User	UserName
Password	Password

Table 2: Tagged values for "MainMenu" component of type Menu

Tag	Value
Id	cpnMainMenu
Navigation	Link("Logout", cpnLogin,,0)
Options	Link("Pending Trips", cpnPendingTrips,,1); Link("My Current Trips", cpnCurrentTrips,,2); Link("My Finished Trips", cpnFinishedTrips,,3);

Table 3: Tagged values for "My Current Trips" component of type List

Tag	Value
Id	cpnCurrentTrips
Navigation	Link("Main Menu", cpnMainMenu,,0)
MainEntity	Trip
FixedFilters	Driver = LOGGEDUSER AND TripStatus in (TripStatus.Accepted, TripStatus.Started)
Columns	CustomerAddress; DestinationAddress; Retrieve(min(EventDateTime), TripLog, TripStatus=TripStatus.Pending, "Request Date"); OptionalLink("Start", cpnStartTrip, "ObjectID = TripID",TripStatus=Accepted AND not Exist (TripStatus.Started));

	OptionalLink("Finish", cpnFinishTrip, "ObjectID = TripID",TripStatus=Started); OptionalLink("Decline", cpnDeclineTrip, "ObjectID = TripID",TripStatus=Accepted);
Sort	"Request Date" ASC

Table 4: Tagged values for "Accept Trips" component of type UpdateView

Tag	Value
Id	cpnAcceptTrip
Navigation	Link("Main Menu", cpnMainMenu,,0); Link("Back", cpnPendingTrips,,9);
MainEntity	Trip
CreateEntity	TripLog
DisplayProperties	CustomerAddress; DestinationAddress; Retrieve(min(EventDateTime),TripLog, TripStatus=TripStatus.Pending,"Request Date"); CustomerPhone
UpdateProperties	TripLog.Remarks
DefaultValuesUpdate	TripStatus = TripStatus.Accepted; Driver = LOGGEDUSER; TripLog.Driver = LOGGEDUSER; TripLog.EventDateTime = NOW

Table 5: Tagged values for "My Finished Trips" component of type Search

Tag	Value
Id	cpnFinishedTrips
Navigation	Link("Main Menu", cpnMainMenu,,0);
MainEntity	Trip
FixedFilters	DriverID=LOGGEDUSER;
Columns	CustomerAddress; DestinationAddress; Price; Retrieve(EventDateTime, TripLog, TripStatus=TripStatus.Finished, "Date"); TripStatus; Link("Trip Log",cpnTripLog, ObjectID=TripID;action=R);
SearchFilters	TripStatus: SingleSelection; CustomerAddress: FreeText; DestinationAdrress: FreeText; Date: DateRange;
Sort	"Date" ASC

Table 6: Tagged values for "View Trip" component of type CRUD

Tag	Value
Id	cpnViewTrip
Navigation	Link("Main Menu", cpnMainMenu,,0); Link("Back", BACK,9); Link("Trip log", cpnTripLog, ObjectID = TripID ,1);
MainEntity	Trip

To complete the model a UML state chart diagram is created for the Trip.TripStatus property to ensure following a valid sequence when updating a Trip

(see figure 5). This is necessary because two or more drivers could try to accept the same trip at the same time but only one should be able to do it.

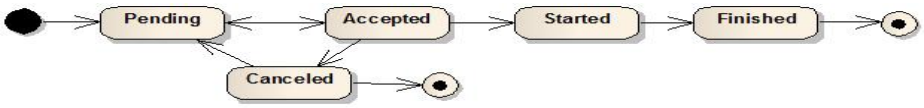

Fig. 5. UML state chart diagram for valid Trip.TripStatus sequence

4 Conclusions and Future Work

This methodology allows modeling applications in an easy way by using just a few diagrams. All diagrams are based on UML and all the proposed extensions are conservative so the modeling could be made with any modeling tool supporting XMI to export the models. By using templates to generate the source code is possible to have different implementations with the same model to target different platforms as it was shown in the example with a mobile web application front end and a standard web application for the backend.

The future work includes:

- Programming the transformation tool and increasing its automation by connecting to the database engine and executing the resulting script and allowing automatic build and deploy of the application.
- Defining different templates for generating source code for different platforms.

5 References

1. Ceri S., Fraternali P., Bongio. "Web Modeling Language (WebML): a modeling language for designing Web sites", Computer Networks, Volume 33, Issues 1–6, (2000), pp 137-157.
2. Cowan D. and Lucena C.. "Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse". IEEE Trans. Softw. Eng. 21, 3 (1995), pp. 229-243.
3. Isakowitz, E. Stohr A. and Balasubramanian P. "RMM: a methodology for structured hypermedia design". ACM (1995), 34-44.
4. Kleppe A., Warmer J., Bast W. "MDA explained: the model driven architecture: practice and promise". Addison-Wesley Professional (2003)
5. Koch, Knapp, Zhang, Baumeister.Uml-Based Web Engineering, Chapter 7 "Web Engineering: Modelling and Implementing Web Applications", Springer London (2008), pp 157-191
6. OMG, "MOF 2 XMI Mapping", Version 2.4.1 (2011), http://www.omg.org/spec/XMI/
7. Pineda C. "Un Método de Desarrollo de Hipermedia Dirigido por Modelos". Tesis Doctoral. Universidad Politécnica de Valencia. (2008)
http://riunet.upv.es/bitstream/handle/10251/3884/tesisUPV2961.pdf
8. Schwabe D. y Rossi G. "An object oriented approach to Web-based applications design". Theor. Pract. Object Syst. Volume 4, Issue 4 (1998), pp 207-225.
9. W3C, "Mobile Web Application Best Practices", 2010, http://www.w3.org/TR/mwabp/
10. W3C, "Mobile Web Best Practices 1.0", 2008, http://www.w3.org/TR/mobile-bp/
11. W3C, "XSL Transformations (XSLT)". Version 1.0 (1999). http://www.w3.org/TR/xsl