Modeling Complex Mobile Web Applications from UI Components – Adding Different Roles and complex Database Design

Pablo Vera[1], Claudia Pons[2], Carina González González[3], Daniel Giulianelli[1], Rocío Rodríguez[1]

[1] National University of La Matanza
Department of Engeniering and Technological Research
San Justo, Buenos Aires, Argentina
{pvera, dgiulian, rrodriguez }@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
[3] La Laguna University
Department of Systems Engineering and Automation,
Architecture and Computer Technology
La Laguna, España
cjgonza@ull.es

Abstract. Component Based Hypermedia Design Method (CBHDM) is a modeling methodology that allows creating mobile web applications by designing and configuring user interface components. Starting from models this methodology performs two transformations to finally generate the application source code using the MDA approach. In order to configure the user interface components this methodology creates a custom language that´s powerful enough for designing complex applications. This paper shows how to configure components for allowing complex database design and also includes a new feature on the model supporting different user roles assigning different screens to operate the system.

Keywords: MDA, Mobile Web Applications, Mobile, UML, User Roles

1 Introduction

MDA (Model Driven Architecture) [1] is an approach for developing systems by building models and generating the application source code automatically or semi automatically by following some transformations steps. In order to be able to model complex systems the modeling methodology must support advanced capabilities like complex queries over the data model and assigning different views to different user roles.

Several methodologies use MDA approach to model web applications starting from the conceptual model and defining the navigational design. Some of them also include the desire capabilities to support advance modeling such as:

- Object Oriented Hypermedia Design Method (OOHDM) [2] allows querying data by using a sql like syntax over the objects. This syntax is used in the class definition to set related properties. For assigning different behavior for different roles a navigational model must be done for each role, defining a view of the conceptual model for each role.
- Web Modeling Language (WEBML) [3] includes an Object Query Syntax for accessing related data. It also includes support for users and groups allowing the definition of pages that will be visible by defining a site view for each group.
- Engineering Web Applications Using Roles [4] discuss role modeling in web engineering and proposed a notation for assigning roles to conceptual and navigation models of different methodologies.
- A MDA Approach for Navigational and User Perspectives [5] models roles with UML actors and hierarchy and then defines zones where those roles can operate. For each zone a navigation diagram is created.

In this paper we present a modeling methodology named "Component Based Hypermedia Design Method (CBHDM)" [6], with new features that allow creating mobile web applications through user interface components. Thus, the paper is organized as following: Section 2 will briefly introduce the methodology. Section 3 will explain the new features added to the methodology for supporting different user roles modeling. Section 4 will explain how to use the configuration capabilities of the methodology to support complex data design and query. Finally section 5 will show the conclusions and future work.

2 Component Based Hypermedia Design Method (CBHDM)

CBHDM is a design methodology for designing mobile web application. It´s based on a conservative extension of UML. It adds some necessary characteristics on class and component diagrams allowing a detailed modeling. Those models will have all necessary information for automatically deploying an application source code by using the MDA approach.

The methodology starts from the UML Class diagram where the system entities are defined using some stereotypes that will be used on the final transformation to facilitate the process of generating the database script, and for identifying and describing entities on the system. Later a transformation tool uses the class diagram to automatically generate a Component Diagram that the designer will modify and complete with the desire behavior of the interface. An additional UML State chart diagram could be used to define object states sequences that later will be checked on the system business logic.

The last step consists in using the transformation tool for a second time with all models as input. The result will be a database script and a fully functional application source code. Figure 1 shows the different stages of the methodology and also remarks

the steps that requires user participation. More details about the methodology can be found in [6].

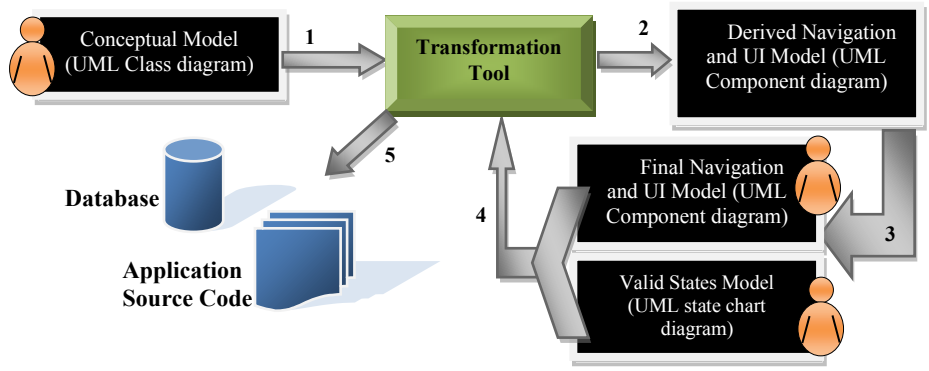

Fig. 1. CBHDM Methodology Stages

The power of CBHDM lies in the ability to configure pre-defined components.

The configuration is performed using tagged values with a detailed semantic for each value. The semantics is declared on a BNF that defines the configuration language.

This configuration language is powerful enough for customizing the components and also for accessing the data present on the conceptual modeling. It includes several functions that allow complex data resolution and query.

3 Roles

In order to allow defining different functionalities according to the role of the logged user a new parameter was added to the link function, the `RoleCondition`. The link function is used to show a link on the user interface to navigate to other component; it's the base of the navigation system and of the `MainMenu` component. A secondary function called `OptionalLink` was present allowing modeling links that are only visible if the condition is accomplished. In order to give more power to the system and to separate concerns the new role condition parameter was added instead of using the condition already present on the `OptionalLink` function giving more configuration power and encouraging clarity. The new parameter was added on both functions, `Link` and `OptionalLink`.

The role condition adds a rule that must be checked to determine the visibility of the link. This allows showing some links only to specific user roles.

The new form of the link function is:

```
<Link>::='Link('<LinkText>','<BrowsableComponent>','
<OptionalLinkParameters>','<OptionalAccessKey>','
<OptionalRoleCondition> ')'
```

And the new form of the OptionalLink function is:

```
<OptionalLink>::='OptionalLink''('<LinkText>','
<BrowsableComponent>','<OptionalLinkParameters>','
<OptionalAccessKey>','<LinkCondition>','
<OptionalRoleCondition>')'
```

The condition will be automatically related with the logged user, so the starting point must be the class representing the logged user. This approach adapts to different ways of representing the role assignment security that are next explained.

3.1 Direct Role on user class.

When role assignment is directly given by a property on the user class with a boolean property like in the example of the figure 2.

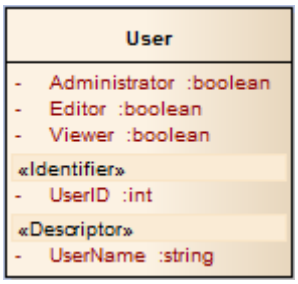

Fig. 2. Class with role assignment as property

A link that must be seen only by an administrator in the example of figure 2 will have the following Role Condition:

```
User.Administrator = true
```

The link can also be assigned to more than one role aggregating conditions, for example the following Role Condition will make the link visible either for administrators or editors users:

```
User.Administrator = true OR User.Editor = true
```

3.2 Unique Role with related class.

If the user has a unique Role the more common approach will be a property on the user class related to the role class. The Role class will establish the different roles on the system by enumeration values. So the designer could refer to those values to configure the access. An example of this approach can be seen in figure 3.

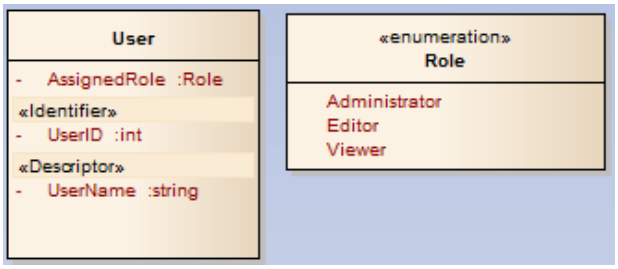

Fig. 3. Unique Role Assignment schema

A role condition restricting the visibility only to the administrator role will be:

```
User.Role = Role.Administrator
```

3.3 Several Roles with related class.

If the user can have more than one role and the roles are on a separate enum class. The conceptual model will have a class to assign security like the example in figure 4.

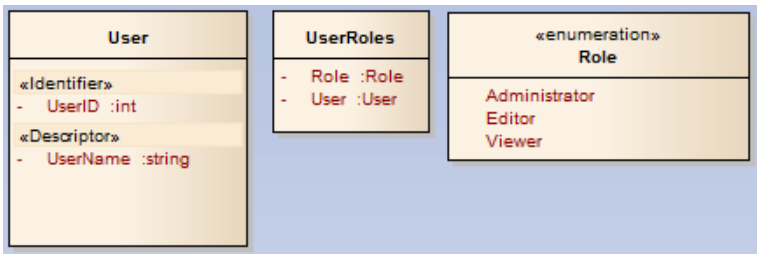

Fig. 4. Separate class for role assignment

In this case the main class of the condition is not the user class, so we must explicitly configure the field relating to the logged user like the code below:

```
UserRoles.User=LOGGEDUSER AND UserRoles.Role=Role.Editor
```

4 Complex Data Base Design

When modeling a system, a good database design is a key point for avoiding redundancy and for obtaining a correct system performance when accessing data.
CBHMD automatically generates the database from the conceptual model, so each class will be transformed on a database table. On each table a number of operations will be carried out to improve performance:

- The primary key will be set in the property marked as Identifier
- An index will be created for each property related to another table, for improving joins

– An unique index will be created for the descriptor property to avoid duplicated values

So the designer must create the class diagram thinking on the database design, knowing that later, data could be accessed due to a powerful configuration language for components. This will allow for example creating log records on a separate table when updating a table and accessing data by complex querying.

In order to illustrate the possibilities of the configuration language some examples will be shown next. All examples will be based on the conceptual model of the figure 5 related to a mobile system for registering trips in a taxi company. Note that CBHDM models class relationships by adding a property with the type of the related entity like in database design.

Fig. 5. Conceptual Model for registering trips on a taxi company

4.1 Complex Querying

CBHDM configuration language allows the use of object notation for accessing data on related classes. But also includes several functions to access indirect information present on the model. Those functions are Sum, Count, Exist, Not Exist, Eval and Retrieve.

Sum.

This function allows obtaining the sum of some property on a related table. For example getting the sum of the prices of trips finished of the logged driver:

```
Sum (Trip.Price, Trip.Driver = LOGGEDUSER AND
Trip.TripStatus = TripStatus.Finished)
```

The first parameter of the function is the property to be summed, the second are the conditions applied prior performing the operation.

Count.

It allows obtaining the quantity of objects that fulfills the condition. For example the following code gets the number of trips finished by the logged driver:

```
Count (Trip, Trip.Driver = LOGGEDUSER AND
Trip.TripStatus = TripStatus.Finished)
```

In this case the first parameter is the entity where the objects must be counted.

Exists and Not Exists.

These functions return a Boolean value to check if any object with a given condition exists. For example the following code creates a link to start a previously accepted trip only if a previous trip was not started before:

```
OptionalLink("Start", cpnStartTrip, "ObjectID =
TripID", TripStatus=TripStatus.Accepted AND not Exist
(TripStatus.Started));
```

The entity where to check the existence is not defined in this case because the link is part a component where the main entity was configured as Trip, so all related operations will be done on the Trip class. Otherwise the full syntax of the condition will be:

```
Trip.TripStatus=TripStatus.Accepted AND not Exist
(Trip.TripStatus.Started)
```

Eval.

It is a function to evaluate a condition and returning a Boolean value. For example in a table if we need to show inside a column that the trip price was 0 (bonus trip) we can use the eval function to check it:

```
Eval(Trip.Price = 0)
```

If the trip price was 0 it will show the “true” word in a column that for example could be labeled as “Bonus Trip”.

Retrieve.

When working with log classes is usually more efficient retrieving information from those classes instead of duplicating data on source class. So the retrieve function will be used to go and get the related data. For example if we want to show the initial request date of the trip in a grid we need to access the log table and find the initial status of the trip:

```
Retrieve (min(EventDateTime),
TripLog.TripStatus=TripStatus.Pending);
```

The first parameter is the property to retrieve and the second the condition. This function must be used in the context of a list where the id of the trip is taken from the row being displayed and this is an implicit filter for the TripLog.Trip field.

4.2 Partially Updating records

In several systems an object class goes through different states, and on each state not all its properties must be able to be updated. For that reason CBHDM adds the `UpdateView` component that was specially created for allowing that partial update

of the object. For example, the driver has a list with available trips and he wants to confirm that he will perform that trip. In that case all trip information is already set and the driver will only change the trip status and eventually fill a text with some remarks. The `UpdateView` component allows showing some properties in only read mode and other in edition mode. In the example of table 1 the remarks are configured to be editable only by the user and the status is automatically changed when updating the object.

Table 1: 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
DisplayProperties	CustomerAddress; DestinationAddress; Retrieve(min(EventDateTime),TripLog, TripStatus=TripStatus.Pending,"Request Date"); CustomerPhone
UpdateProperties	TripLog.Remarks
DefaultValuesUpdate	TripStatus = TripStatus.Accepted; Driver = LOGGEDUSER;

4.3 Creating additional records

Usually when an object is created or updated a new record in a related table must be created. This approach is very common for a log class that records the changes made on some particular class. CBHDM includes a special tagged value called `CreateEntity` that creates an object on the entity configured on the value of this tag. For example, if the system must keep record of the change of the status of the trip that was configured on table 1, the `CreateEntity` tag can be used. Table 2 shows the complete component configuration adding a configuring log record creation. The values of the newly created object are configured with object notation as can be seen in the value of the `DefaultValuesUpdate` tag of table 2.

Table 2: Tagged values for "Accept Trips“ component of type UpdateView with log record

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`

For the same purpose two tagged values were added to the CRUD component that allows creating and updating class objects: `CreateEntityOnCreate` and `CreateEntityOnUpdate`. This allows adding related records when creating an object or when updating if `UpdateView` component is not used.

5 Conclusions and Future Work

CBHDM allows modeling systems with all necessary information for automatic code generation. The use of functions to retrieve data is powerful enough for accessing related records and for complex querying as shown in the examples above.

The new role property on links allows the designer to assign tasks to each role for increasing system security and to assign responsibilities without the need of creating a separate model for each role like the methodologies explained in section 1.

Finally the ability to create related records when updating or creating a main class is an essential characteristic that completes the model and allows for example keeping track of modification in a separate log table.

CBHDM methodology and language configuration is now completely defined and the transformation tool is being developed. Future work consists on finishing the mentioned development and validating the results by performing modeling of different complexity.

6 References

1. Kleppe A., Warmer J., Bast W. “MDA explained: the model driven architecture: practice and promise”. Addison-Wesley Professional (2003)
2. 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.
3. 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.
4. Rossi G. Nanard J., Nanard M and Koch Nora, “Engineering Web Applications Using Roles“, Journal of Web Engineering, Vol. 6, No.1 (2006
5. Gonzales M, Casariego J., Bareir J., Cernuzzi L, Pastor O. “A MDA Approach for Navigational and User Perspectives”, Special issue of best papers presented at CLEI 2010 (2011)
6. Vera P., Pons C. Gonzales C, Giulianelli D., Rodriguez R. “MDA based Hypermedia Modeling Methodology using reusable components”, XVIII Congreso Argentino de Ciencias de la Computación (2012)