A navigational role-centric model oriented web approach – MoWebA

Magalí González* and Luca Cernuzzi

Catholic University of Asuncion,
Asuncion, Paraguay
Email: mgonzalez@uca.edu.py
Email: lcernuzz@uca.edu.py
*Corresponding author

Oscar Pastor

Research Center on Software Production Methods,
Universitat Politècnica de Vàlencia,
Valencia, Spain
Email: opastor@pros.upv.es

Abstract: This study presents MoWebA, a navigational role-centric model driven development (MDD) proposal for web applications development. The approach was conceived considering a previous study of web methods and analysing some open issues. This article presents the fundamentals of the proposal; the methodological aspects for modelling and transformation processes; and the defined notations/techniques for modelling and transformation tasks, including their abstract and concrete syntax definitions. We include a summary of the validation experiences and main results, and a comparison against other related proposals, in order to highlight the main contributions of MoWebA.

Keywords: model driven architecture; MDA; model driven development; MDD; web application; web methodologies; navigational models.

Reference to this paper should be made as follows: González, M., Cernuzzi, L. and Pastor, O. (2016) 'A navigational role-centric model oriented web approach – MoWebA', *Int. J. Web Engineering and Technology*, Vol. 11, No. 1, pp.29–67.

Biographical notes: Magalí González obtained her Informatics Engineering degree from Catholic University of Asuncion, Paraguay in 2000, and currently she is a PhD student at Polytechnic University of Valencia, Spain. Since 2002, she has been a Professor in the Department of Electronics and Computer Science Engineering (DEI in Spanish) at the Universidad Católica Nuestra Señora de la Asunción – UC (Asunción, Paraguay). Currently, she is an Associate Professor in Model Driven Engineering discipline. Her current research interests include: software engineering, model driven engineering, web engineering, among others. In these areas, she has published over 20 papers in journals, book chapters, and international conferences and workshops.

Copyright © 2016 Inderscience Enterprises Ltd.

Luca Cernuzzi obtained his Laurea degree in Computer Science from University of Milan in 1990, and PhD in Engineering from University of Modena e Reggio Emilia in 2007. Since 2000, he has been a Full Professor in Software Engineering at the DEI – 'Universidad Católica Nuestra Señora de la Asunción' (Paraguay). He has lectured as a Visiting Professor at several European and Latin American universities. His current research interests include: software engineering, web engineering, data science, ICT for good, and social informatics. In those disciplines, he has published over 100 papers in journals, book chapters, and international conferences and workshops.

Oscar Pastor is a Full Professor and Director of the 'Centro de Investigación en Métodos de Producción de Software (PROS)' at the Universidad Politécnica de Valencia (Spain). He is the Chair of the ER Steering Committee (2009–2010), ER Fellow since 2010, member of the SC of conferences such as CAiSE, ER, ICWE, ESEM, CIbSE or RCIS, and member of over 100 scientific committees of top-ranked international conferences. His research activities focus on conceptual modelling, web engineering, requirements engineering, information systems, and model-based software production. He created the object-oriented, formal specification language OASIS and the corresponding software production method OO-METHOD. He led the research and development underlying CARE Technologies that was formed in 1996.

This paper is a revised and expanded version of a paper entitled 'Una Aproximación para Aplicaciones Web: MOWEBA' presented at XV Iberoamerican Conference on Software Engineering – CibSE, Rio de Janeiro, Brazil, 28 April 2011.

1 Introduction and motivation

Web development has motivated the so-called 'web engineering' (Deshpande et al., 2002; Pressman and Lowe, 2009), which focuses on methodological web proposals, in order to improve the quality of the web development process and the final product. Current web methods centre on developing techniques and/or models needed to define the design processes, and on providing tools to support them (Mernik et al., 2005), following the model driven development (MDD) approach in many cases (Brambilla et al., 2012). Some methods have tool support for generating automatic prototypes [e.g., VisualWADE for OO-H (Gómez et al., 2005)], but only a few, such as WebRatio for WebML, have automation tools tested in industrial settings. There are various quantitative and qualitative studies that show how MDD practices contribute to increase the efficiency and effectiveness in software development (Acerbis et al., 2007; Blechar and Norton, 2009).

The study of web methods and the classification proposed by Schwinger and Koch (2006), as well as our previous experiences and that of different authors (De Troyer and Decruyenaere, 1998; Cachero and Koch, 2002; Bozzon et al., 2006; Meliá et al., 2005), reveal some concerns. Below we list those more important from our point of view.

The *first concern* establishes that 'navigational oriented modelling could help simplify the models for web applications'. Navigation has been identified as a critical and fundamental feature within web engineering (Rossi et al., 2007; De Troyer and Casteleyn, 2003). Nevertheless, navigational models are usually not the starting point of the modelling process. In some situations, navigational models do not provide an appropriate syntax to model common behaviours of current web systems, such as the dynamic navigation behaviour observed during users' interaction, or inter-intra contextual navigation. Most of the methodologies mentioned in the literature [UWE (Koch et al., 2007), WebML (Ceri et al., 2000), OOWS (Fons et al., 2007), OO-H (Cachero et al., 2000), OOHDM (Schwabe and Rossi, 1998)] start the design of navigational models from the conceptual (i.e., structural) model. However, the way in which the information is arranged and structured in the organisation, is not necessarily the way external users need to access it (De Troyer and Decruyenaere, 1998). Thus, deriving the navigational model from the structural model may be useful in order to organise the information content, but this does not model users' interaction in all their dimensions. Modelling the navigational perspective according to the way in which user wishes to explore the application (i.e., functional-oriented modelling) helps to obtain friendly and easy to access navigational paths. Therefore, the open issue is to find alternative ways to model the navigational perspective better fitting the requirements of users' interaction and making user navigation more adherent to its mental model.

A *second concern* is that the "adoption of standards will facilitate interoperability between models, methods, transformations rules, and tools". In recent years, methodologies such as UWE (Koch et al., 2007), WebML (Ceri et al., 2000), W2000 (Baresi et al., 2006), OOWS (Fons et al., 2007); and tools such as Acceleo (http://www.acceleo.org), AndroMDA (http://www.andromda.org), Olivanova (http://www.sosyinc.com), Optimal J (http://www.compuware.com), ArcStyler (http://www.markosweb.com/www/arcstyler.com), among others, have partially adapted their models, processes and/or transformation languages to the model driven architecture – MDA (*MDA Guide Version 1.0.1*, http://www.omg.org); MDA propose using several standard languages to follow MDD. Without adopting MDA approach in all its potential, the methodologies tend not to take advantage of the efficiency and effectiveness in web engineering. Despite UWE being the only methodology whose models and processes completely follow the MDA approach, their code generation tools require additional adjustments for a complete transformation (e.g., UWE4JSF which works in the eclipse environment and generates JSF applications requiring additional adjustments for some java classes, libraries, style sheets, among others). For the semi-automatic generation of web applications some other approaches were implemented and are currently under evaluation (http://uwe.pst.ifi.lmu.de/). In any case, it is an open line of research how to take profit from the adoption of standards, transformation tools, and the thorough MDA potential in web engineering.

Finally, the *third concern* is the belief that "taking into account evolution of web environments is very important for improving the development of current web applications". In fact, current web applications evolve very fast (considering technologies, platforms, architectures, diversity access devices, among others) and methodologies need to be flexible in order to consider these web tendencies. Normally, methodologies try to do this by extending their modelling notations [e.g., RIAs proposal for WebML (Bozzon et al., 2006)] at the level of platform independent model (PIM). In doing so, the PIMs are not technology/platform independent anymore, and they are becoming increasingly complex to understand and manage. The consequence is a loss of portability of the models. Therefore, the open issue is to find alternative ways to assure the easy evolution of web application as well as preserving the independence of the PIM and the portability of models for different platforms.

Model oriented web approach (MoWebA) try to respond to the previous concerns and their related open issues. It adopts the MDD approach in every phase and the corresponding supporting tools trying to offer more efficiency and effectiveness in web applications development; it offers an innovative proposal for the navigational perspective; and it considers the new technological tendencies in web applications.

The main contributions of MoWebA are:

1 providing a view of navigation, more function-oriented (i.e., behavioural-oriented) than data-oriented, trying to better capturing the requirements of users interaction

2 considering almost all the modelling process, starting from the navigational model instead of the conceptual/data model

3 providing an architectural level of modelling definition titled ASM – architectural specific model, in order to facilitate the evolution of applications. In this study, we present the dimensions and the processes of MoWebA and its use in different experiences paving the way for a more rigorous validation of the proposal.

The rest of the article is organised as follows: Section 2 presents a general overview of the MoWebA proposal; Section 3 presents the MoWebA modelling process; Section 4 includes the MoWebA transformation process; Section 5 explains some experiences of MoWebA; Section 6 presents related works. Finally, we present the conclusions and future works in Section 7.

2 The model oriented web approach – MoWebA

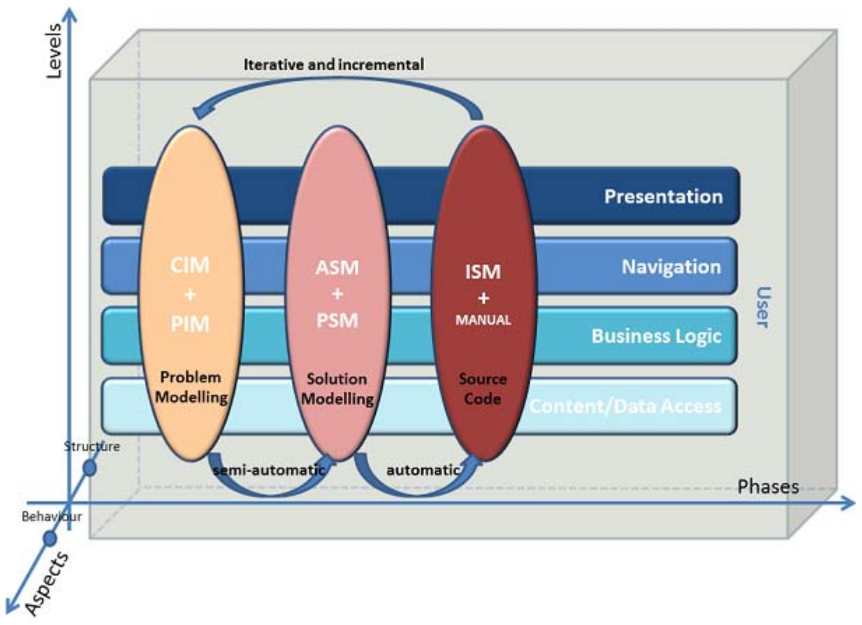
MoWebA defines methodological aspects (processes, stages, work products, dimensions) and complements these aspects with an entire environment, including modelling and transformation tools, automatic code generation, use of standards, and layered architecture, among others. For this reason, we refer to MoWebA as a 'navigational role-centric model-based approach to web application development'.

Figure 1 shows the MoWebA dimensions: phases, levels and aspects.

The *phases dimension* covers the modelling and transformation processes. MoWebA adopts the MDA approach by identifying three different abstractions for modelling: the problem space, covered by computational independent model (CIM) and PIM models; the solution modelling space, covered by architectural specific model (ASM) and platform specific model (PSM); and the source code definition, covered by implementation specific model (ISM) and manual code. The *levels dimension* deals with complementary perspectives to be considered in every phase (content, business logic, navigation, presentation, users). Finally, the *aspects dimension* addresses the structure and behaviour considerations for each perspective.

MoWebA defines two main complementary processes: one related to the modelling activities and the other to the transformation activities. As shown in Figure 1, the horizontal axis represents the MoWebA transformation process. To formalise the modelling and transformation processes, it adopts the MOF language for abstract syntax definition, and the UML profile extension for a precise definition of the modelling language.

Figure 1 MoWebA dimensions (see online version for colours)



The modelling process includes the necessary activities to get all the diagrams for the complete specification of the system-to-be (considering the problem space, architecture/s, and destination platform/s). This process considers the CIM, the PIM, the ASM and the PSM with their corresponding modelling activities. CIM definition covers the late requirements identification, focusing on functional requirements specifications. PIM specification is based on five models, offering a strong separation of concerns: domain, logic, navigation, presentation, and user. The ASM enriches the models with information for a specific architecture [e.g., rich internet applications (RIAs), service oriented applications, REST, among others] and the PSM contemplates information for a target platform (e.g., a specific language, or a framework).

The transformation process, on the other hand, is related to the steps, techniques, and tools, which allow M2M (i.e., model-to-model) and/or M2T (i.e., model-to-code) transformations. This process is based on the MDA approach, and implies steps and activities for transforming specification in order to go through each MoWebA phase (i.e., CIM/PIM-ASM/PSM, ASM/PSM-ISM/Manual adjustments). The CIM/PIM-ASM/PSM transformation is done in a semi-automatic way (i.e., introducing some manual adjustments), by defining the metamodels for specific architecture or platform, and the corresponding mapping rules for PIM-ASM/PSM transformations. The ASM/PSM-ISM transformation corresponds to the automatic transformation from the models to the application code. Since real experiences have shown that sometimes manual adjustments are necessary, we consider a 'manual adjustment' phase, where additional code can be added to adapt the application. Finally, the transformation process is done iteratively, allowing an incremental application development.

The next sections detail the modelling and transformation processes of MoWebA.

3 MoWebA modelling process

This section starts by presenting a general overview of the stages and activities, and then going into details for each stage, considering diagrams, notations and tasks involved. To clarify the proposal, we use as an example a web-based academic system. The system supports teachers, students, staff and the general public, and covers a range of basic functions such as: student registrations processing, courses monitoring, and school, department and career management. Teachers have sufficient privileges to manage the courses they are in charge of and provide students with information regarding their current status. Students have the required privileges to track the courses they are enrolled in and also access their current academic status. Finally, the system should provide the facility to perform administrative tasks such as faculty, course, department, and subject management.

The modelling process includes the CIM, PIM, ASM and PSM specification and systematised in seven stages (see Figure 3).

Stages 1 through 6 are oriented to CIM and PIM definitions, based on the dependency relationships between the different models, the level of granularity of the modelling task, and the type of modelling to be done; these stages are done manually. MoWebA adopts the use case model for CIM definition, focusing on modelling the functional requirements of the system-to-be. For PIM definition, MoWebA proposes the following models:

1 entity model

2 navigational model

3 behavioural model

4 presentation model

5 user model.

Each model is composed of one or more diagrams. Figure 2 presents the dependency relationships between the different models.

Stage 1 is related to the requirements analysis. The artefact produced in this stage is a *use case diagram* representing the functional, navigational and usability requirements, as well as potential users of the application. Stage 2 corresponds to the navigational structure, role and domain definition. In this stage a *navigational tree diagram* is defined to organise the system basic functionalities in a hierarchical way. The *role and zone diagrams* are created considering the potential users identified at stage 1. An *entity diagram* defines the structure and the static relationships between classes identified in the problem domain. Stage 3 defines the navigational behaviour for each node through the *node diagram*. Stage 4 defines which elements are going to be displayed on every presentation page using the *content diagram*. The pages structure (positions of headers, menus, footers, among others) is also defined through the *structure diagram*. In addition, structural composition of business process and transactional procedures are defined with the *logic diagram*. In Stage 5 the main activity is to personalise the models through the *adaptation model*. MoWebA proposes *source and rules diagrams* to model different kinds of adaptations (i.e., adaptive). Stage 6 proposes a detailed definition of each service or action identified at *logic and content diagrams* using the *service diagram*.

Figure 2 Diagrams in MoWebA (see online version for colours)

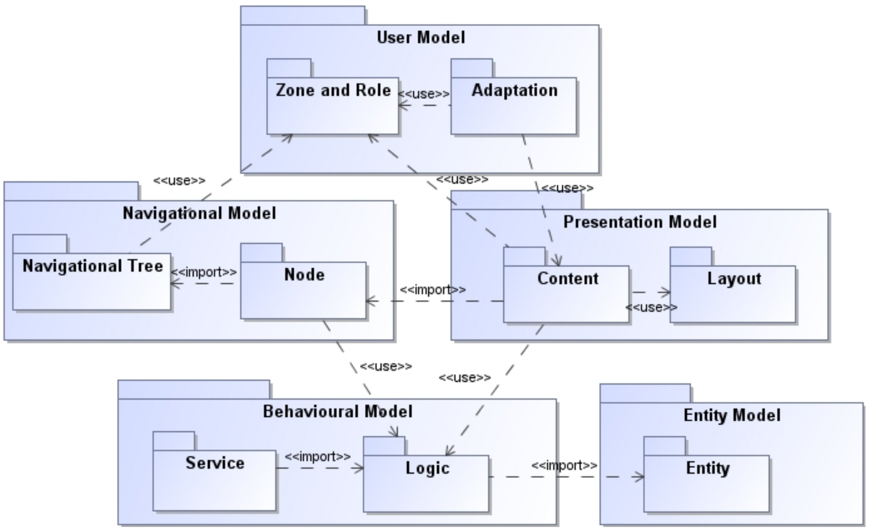
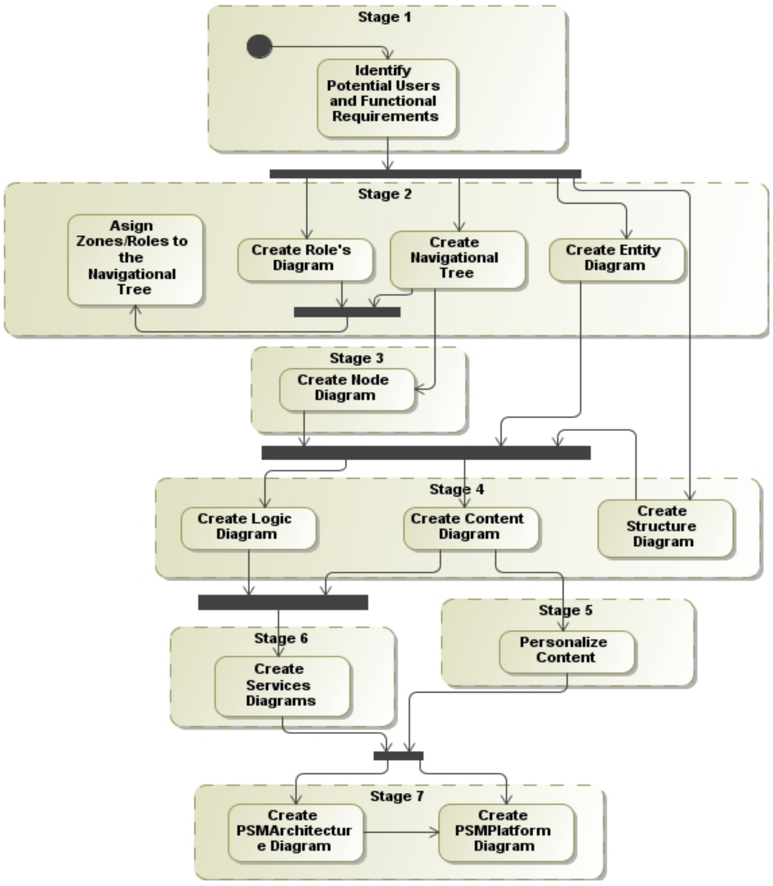


Figure 3 Modelling process (see online version for colours)



Stage 7 contemplates the architectural and platform aspects. This stage is done in a semi-automatic way. It proposes an enrichment of existing models in order to consider aspects related to the final architecture of the system (e.g., RIAs, SOAs, REST), specifying the *ASM diagram*. The next step proposes to add platform specific information (e.g., Ruby on Rails, Python, PHP, Java), specifying the *PSM diagrams*.

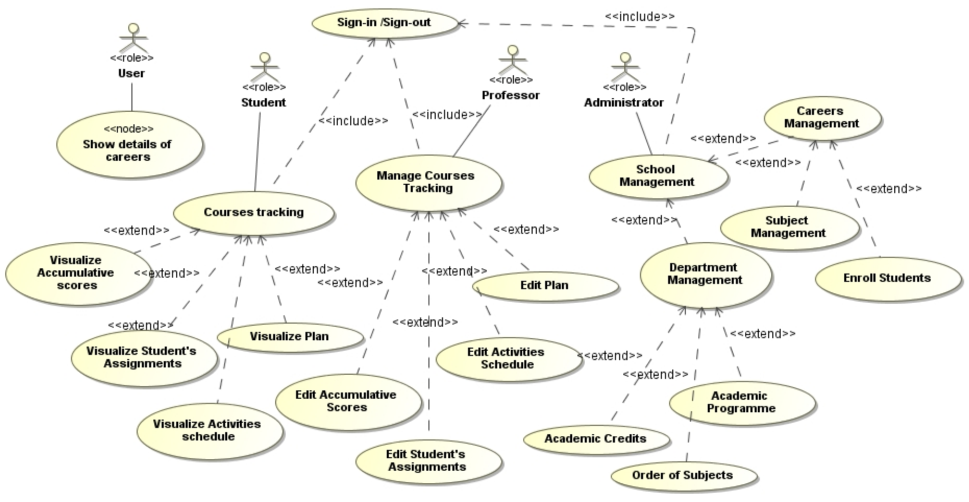
The modelling process is an iterative and incremental process, allowing for diagram refinement. Next sub-sections describe the different stages of the modelling process.

3.1 Stage 1: identify potential users and functional requirements

As a first stage, we need to specify the main goal of the system. In the example, the main goal could be stated as follow: 'to develop a web-based application for academic management of a university in order to process student registrations, course monitoring, and school, department and career management; oriented to students, professors and administrators'.

Early requirements are out of the scope of MoWebA. However, we assume that the designer may use specification scenario-based techniques that already exist in order to get a good understanding of the problem domain (Nuseibeh and Easterbrook, 2000). MoWebA covers the use case diagram with the identification of the different actors and a list of functions associated to the actors (see Figure 4).

Figure 4 The use case diagram for the academic system (see online version for colours)



In this classification, there are some similar or common functions that should be re-organised or re-grouped. In the next stage, we will refine the potential users, identify the domain model and define a navigational structure based on the functionalities defined in this stage.

3.2 Stage 2: specify navigational structure, user roles and domain

This stage defines the following artefacts: *navigational tree*, *role-zone* and *entity*.

Navigation in MoWebA covers both structural and behavioural aspects. The structural aspects are modelled in this stage in terms of 'navigable nodes' and their relationships. A 'navigable node' is a functional unit of the system, and the navigation is 'the change from one navigational node to another as a result of an invocation from the user or an external agent'. Therefore, navigation occurs when an external agent interacts through the invocation of a 'navigational node'.

The *navigational tree diagram* represents the application's navigational space and it is composed of zero or more navigational elements. These elements may be nodes or links. A navigational node connects to other nodes by means of relationships, called hard links, which denote a hierarchy in the navigational tree. The navigational tree is defined following four activities:

1 analyse the use cases defined at stage 1

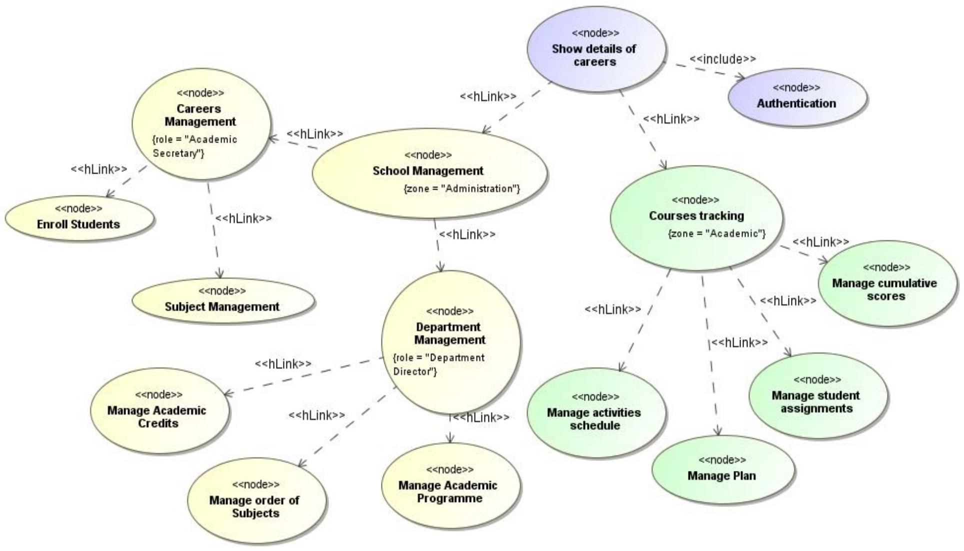
2 analyse the actors diagram for a functional unit hierarchy definition

3 define an initial point for the hierarchical structure

4 create a structure considering the relationships between use cases and actors.

Figure 5 shows an example of a navigational tree.

Figure 5 Navigational tree for the web-based academic system (see online version for colours)


<<node>> Show details of careers
<<include>>
<<node>> Authentication
<<hLink>>
<<node>> Careers Management {role = "Academic Secretary"}
<<node>> School Management {zone = "Administration"}
<<node>> Enroll Students
<<node>> Subject Management
<<node>> Courses tracking {zone = "Academic"}
<<node>> Manage cumulative scores
<<node>> Department Management {role = "Department Director"}
<<node>> Manage Academic Credits
<<node>> Manage activities schedule
<<node>> Manage student assignments
<<node>> Manage Plan
<<node>> Manage order of Subjects
<<node>> Manage Academic Programme

The *navigational tree* has remarkable differences with other approaches in the fundamental concept of the 'navigable node'. The most mentioned methodologies in the literature create the navigational structure from the conceptual model. This has two important implications:

1 the level of granularity of navigational elements are directly related to structural elements (e.g., classes)

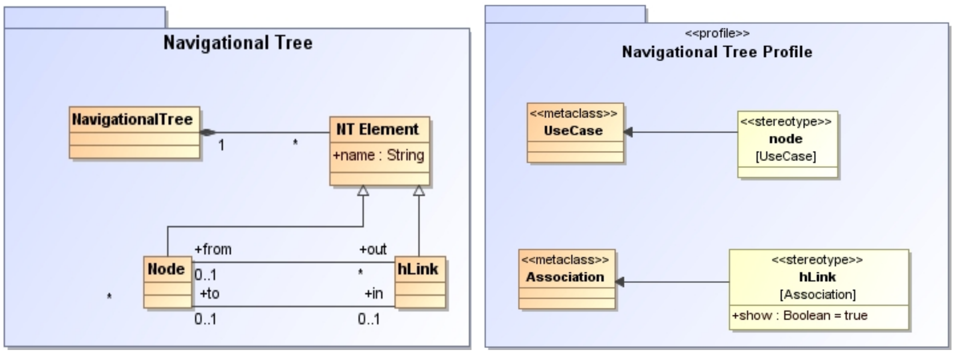
2 navigation is obtained considering the way information is structured (e.g., classes relationships), not the way it is accessed.

In the case of MoWebA, navigation structure is defined considering the functional units as the granularity level, and navigation paths are defined considering hard links between the units, defining though the navigation from the way users interact with the system. With this approach it is possible to model a functional-oriented navigational structure, and to generate several exploration levels, which represent menus and sub-menus, keeping the user located by using 'breadcrumbs' and 'history of navigation'.

However, hard links are not sufficient to specify the navigational structure of an application, because there are situations in which navigation through a different context will be necessary (e.g., once authenticated, the user must specify the destination node). To meet this need, we define the *softLink*, which will be specified in the *node diagram* (Section 3.3).

To formalise the modelling and transformation processes, we used the MOF language for the abstract syntax definition, and UML profile extension for the concrete syntax of the modelling language. The MOF definition specifies MoWebA in terms of a metamodelling language, allowing the definition of concepts in a more rigorous way. Figure 6 shows the navigational tree metamodel and the corresponding UML profile. In this case, only two stereotypes (<<node>> and <<hLink>>) are necessary.

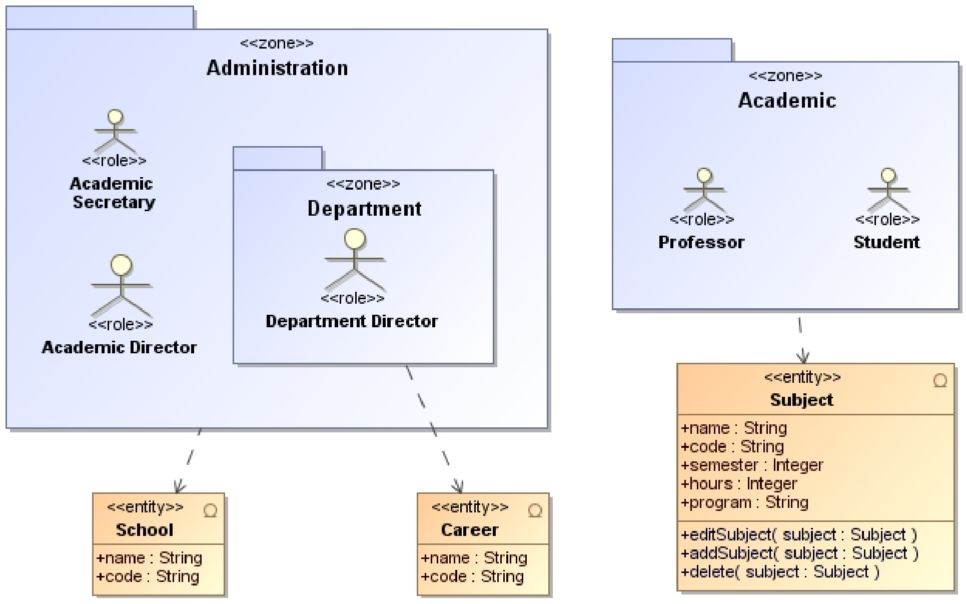
Figure 6 Navigational tree metamodel and UML profile (see online version for colours)



The *role diagram* represents the hierarchy of user roles, that is, groups of users that can access the same functionalities. For this diagram MoWebA proposes the use of the UML actors stereotyped with <<role>>.

The *zone diagram* represents contexts containing certain behavioural profiles in relation to each other. The zones provide system designers the possibility to explicitly define different contexts with multiple roles assumed by users. There may be several zones defined in a system, each one accessed by several roles, and, in turn, users could have more than one role. For example we define a zone in which both students and teachers can access (e.g., subjects or career) and, a different zone for managers (e.g., department). Moreover, the zone could be relative, that is, dependent on a domain class indicating that for a user to assume a certain role, additional information is needed (e.g., at the 'academic' zone, which is accessed by professor and student roles, each user would take at most one of these roles for each subject; see Figure 7).

Figure 7 Example of zone diagram (see online version for colours)



To complete the role and zone modelling task, it is necessary to define roles/zones access privileges on the elements of the system by establishing a dependency relationship between a <<role>> or a <<zone>> and elements of another diagram (i.e., nodes access privileges in navigational tree diagram). A relationship implies that the elements are available for the specified role/zones assigned. Such relationships would be refined in the next stages of other diagrams (logic, presentation, among others). In Figure 5 the node 'course tracking' has privileged access to the 'academic' zone, indicating that both students and teachers have access to that node. The same privileges are inherited by the nodes below in the hierarchy, maintaining access restricted to students and teachers.

Figure 8 presents the zone and role metamodel and UML profile. A role diagram is composed of one or more *RD elements*, which could be specialised in 'user', 'role' and 'zone'. Each *zone* can be composed in one or more *roles* which could have *attributeRoles*. The *zones* could be aggregated by other *zones*, and *roles* can be defined in a hierarchy.

For the *entity diagram* definition, MoWebA adopts the UML class diagram, where each class is stereotyped with <<entity>>. Entities, attributes and relationships are identified by the functionalities description of stage 1.

A simple example of an entity diagram is shown in Figure 9.

Figure 10 presents entity metamodel and UML Profile that includes a new stereotype (<<entity>>).

Figure 8 Zone and role metamodel and UML profile (see online version for colours)

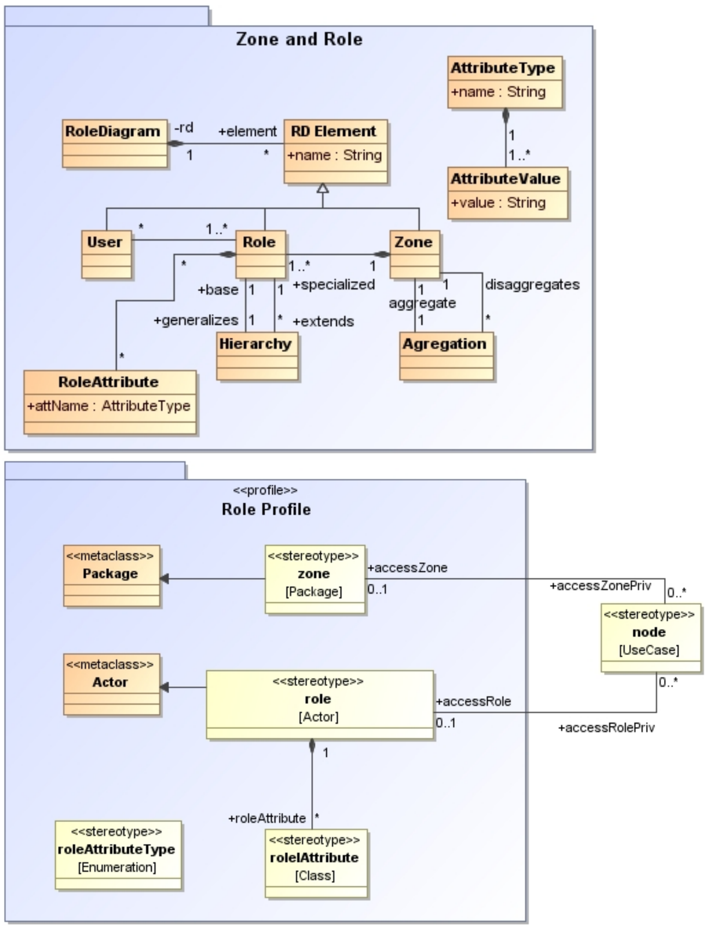


Figure 9 Simplified entity diagram for the web-based academic system (see online version for colours)

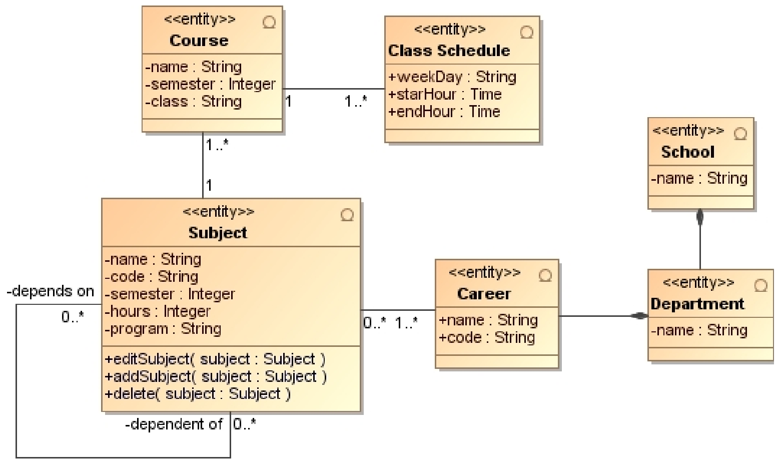
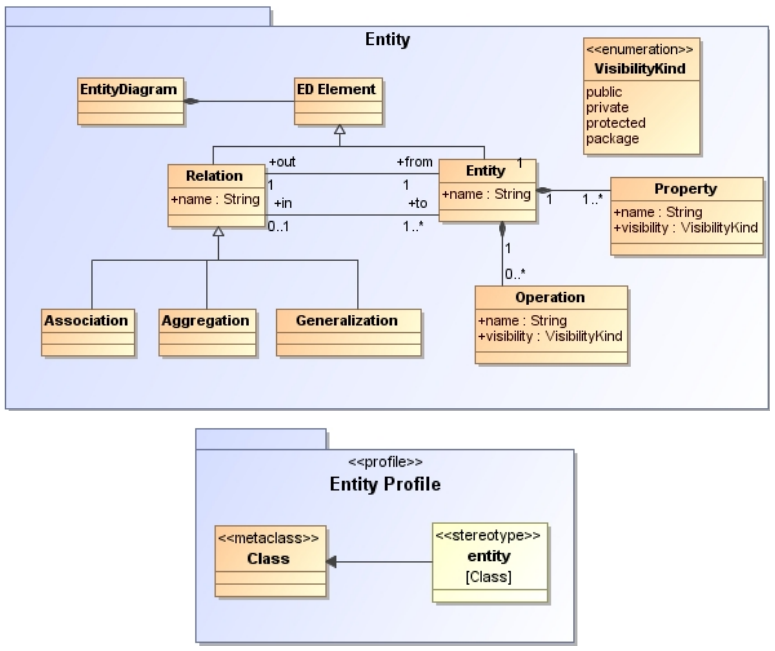


Figure 10 Entity diagram metamodel and UML profile (see online version for colours)



3.3 *Stage 3: specify navigational behaviour*

Each node in the navigational tree must have an associated *navigational node diagram* representing its navigational behaviour. The node diagram is defined using the UML State diagram.

There are three categories of states: flow states, virtual states and final states. Flow states are transient and as such, they are visited only momentarily to create linkages with other elements of the diagram. Flow states can be further classified into four types:

1 initial states

2 pseudo states

3 junctions

4 and service states, which model the services provided by the node.

Virtual states represent stationary states indicating the fact that the navigation flow remains in a 'virtual point' within a node, waiting for an interaction from an external agent. In stage 4, each virtual state will be linked to a presentation page.

The transitions between two states (o state nodes) are specialised in two sub-types: the control flow transitions and the hyperlinks. The control flow models the natural control transfer that occurs between two states, without requiring an external user interaction. The hyperlink models a transition between two states resulting from an invocation of an internal link, which leads to an interaction between the user and the system. A control flow transition can only have a flow state as source, and any type of state as target (e.g., the transition between the service 'login' and 'error message'). The hyperlink transition can only have a virtual state as source, and any state as target (e.g., the transition between ‘entering data’ and service ‘login’). Hyperlinks defined in the node diagram correspond to possible internal navigations, triggered by user interactions. The final state can be connected to another node in the navigational tree; if there is such linkage, it defines a soft link (sLink). This will allow navigation to a unit not directly linked to the functional node of the navigational tree structure.

Figure 11 Node diagram for the authentication tree node (see online version for colours)

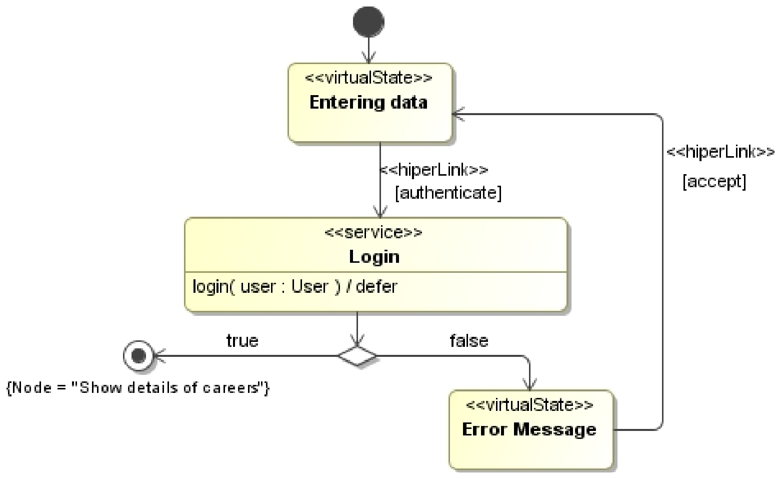


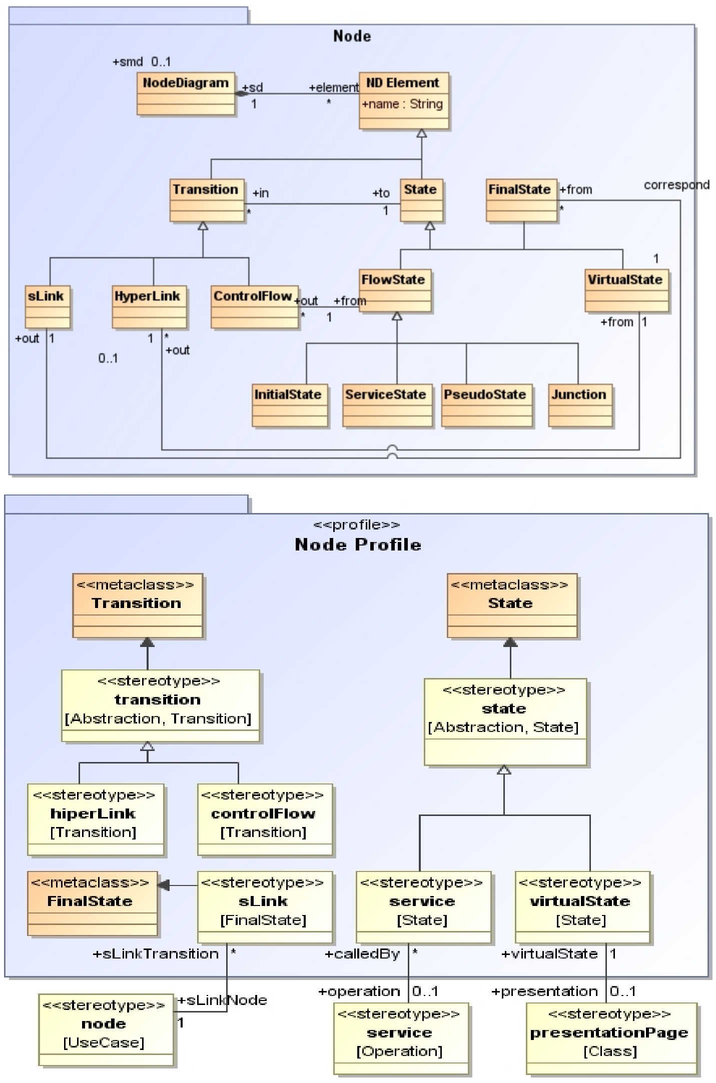
Figure 11 shows an example for the authentication process in which the user has to type a user name and a password (‘entering data’ *virtualState*), then a login service is executed to validate data, and finally, depending on the results, an error message will appear (‘error message’ *virtualState*) or a soft link will take the user to the root node of the system (‘sLinkNode = show details of careers’).

The node diagram allows modelling navigational behaviour aspects obtained from dynamic interactions with the user.

As shown in Figure 12, a node diagram is composed of *ND elements* (node diagram elements). The *ND elements* are classified into *state* and *transition*. *States* in turn are specialised into *FlowState*, *FinalState* and *VirtualState*. On the other hand, *transitions* can be classified as *sLink*, *HyperLink*, or *ControlFlow*. Finally, a number of relationships between the elements have been defined indicating associations that must be considered in order to comply with the different proposed constraints.

In the corresponding UML profile definition, it is possible to notice that the <<state>> stereotype is an extension of the state UML metaclass. The <<transition>> stereotype is an extension of the transition UML metaclass, and the <<sLink>> stereotype is an extension of the *FinalState* UML metaclass. This figure also shows that <<virtualState>> and <<service>> are specialisations of <<state>>, and <<transition>> is specialised in <<hyperlink>> and <<controlFlow>>. Finally, the association between <<virtualState>> and <<presentationPage>> establishes that for each <<virtualState>> of the *node diagram* there should be a <<presentationPage>>. The association between <<sLink>> and <<node>> allows modellers to link a destination node to a final state in the *node diagram*.

Figure 12 Node diagram metamodel and UML profile (see online version for colours)

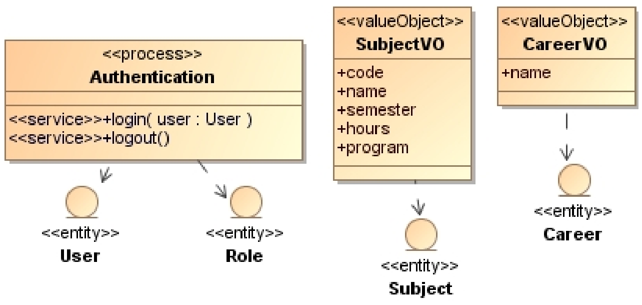


Node
NodeDiagram
ND Element
+name : String
Transition
State
FinalState
sLink
HyperLink
ControlFlow
FlowState
VirtualState
InitialState
ServiceState
PseudoState
Junction
<<profile>>
Node Profile
<<metaclass>> Transition
<<metaclass>> State
<<stereotype>> transition [Abstraction, Transition]
<<stereotype>> state [Abstraction, State]
<<stereotype>> hiperLink [Transition]
<<stereotype>> controlFlow [Transition]
<<metaclass>> FinalState
<<stereotype>> sLink [FinalState]
<<stereotype>> service [State]
<<stereotype>> virtualState [State]
<<stereotype>> node [UseCase]
<<stereotype>> service [Operation]
<<stereotype>> presentationPage [Class]

3.4 *Stage 4: specify logic behaviour and presentation*

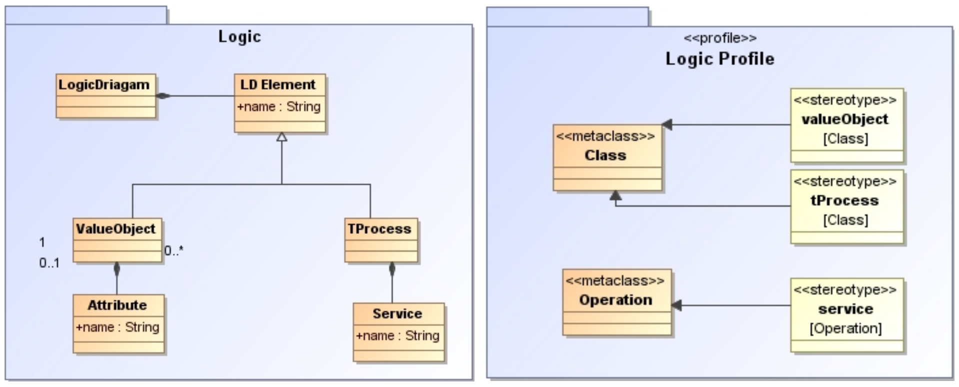
To consider the behavioural modelling, MoWebA defines two diagrams: *logic behaviour and service diagrams*. The logic behaviour diagram encapsulates and structures all the behaviour actions (business processes and transactional procedures) that affect the system. This is done by defining classes stereotyped with <<process>> and <<valueObjects>>. The 'process' class encapsulates business processes that represent complex transactions and are associated through a dependent relationship with one or more classes of the *entity diagram*. These dependency relationships imply that the partners are accessed by the operations defined in the process. On the other hand, the 'valueObjects' class encapsulates data, and depends on one or more entities, containing a subset of attributes defined in the dependent classes. Every service identified in other diagrams, should also be included into the *logic behaviour diagram* as a service for some process. Furthermore, value objects provide domain visibility to the presentation layer. This means that access to the domain has to be done by appropriate value objects defined at the logic behaviour layer. The other behavioural diagram, called *service diagram*, will be explained in stage 6.

Figure 13 Logic behaviour diagram (see online version for colours)



A simplified example of logic behaviour diagram is shown in Figure 13 representing a logic process called 'authentication' which is conformed of two services (login, logout). It is important to notice that the 'login' service has been already defined at the navigational node diagram 'authentication' (see Figure 11). In Figure 13, we define two <<valueObject>> elements, SubjectVO and CareerVO. Notice the dependency between entities and value objects (e.g., SubjectVO and the subject entity).

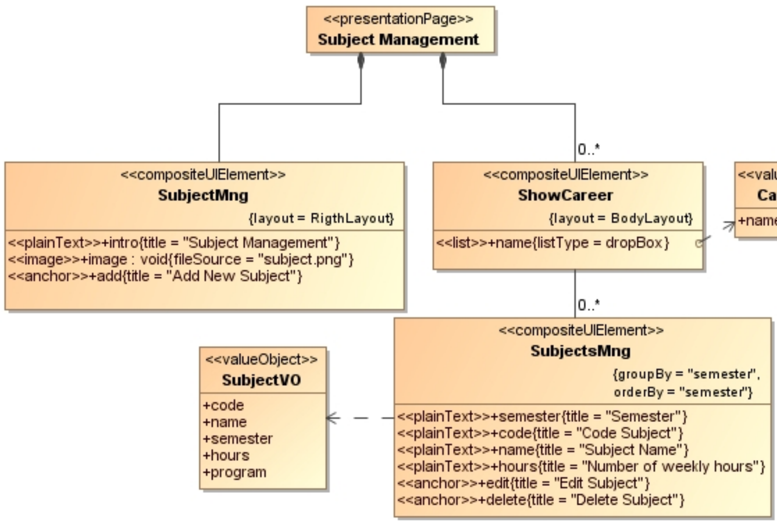
Figure 14 Logic behaviour metamodel and UML profile (see online version for colours)



The *LD elements* of the *logic behaviour* metamodel (see Figure 14) are classified into *ValueObjects* and *TProcess*. The *ValueObjects* are composed of *attributes,* and the *TProcess* of *services* which can be defined in other diagrams (e.g., *services* defined in the node diagram).

The presentation is mainly aimed to facilitate the interaction with the outside world and to provide the necessary elements for users to successfully perform tasks, such as entering data, enabling processes and browsing. For the *presentation model*, MoWebA considers the following aspects: the presentation content; the presentation structure; the format of elements within each region; and the elements' style. Thus, MoWebA defines two presentation diagrams: *content and structure diagrams*.

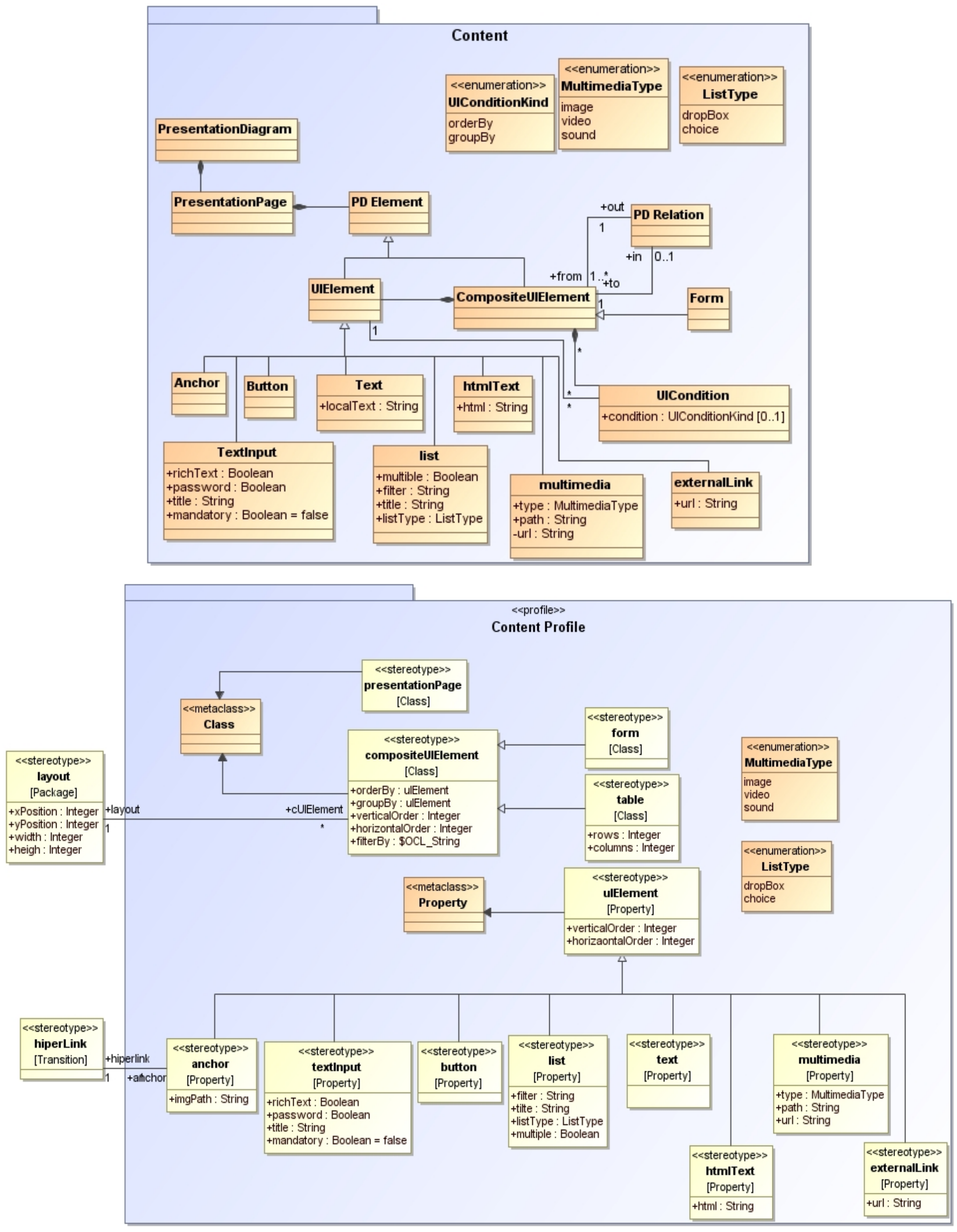
Figure 15 Subject management presentation page (see online version for colours)


<<presentationPage>>
Subject Management
0..*
<<compositeUIElement>>
SubjectMng
{layout = RigthLayout}
<<plainText>>+intro{title = "Subject Management"}
<<image>>+image : void{fileSource = "subject.png"}
<<anchor>>+add{title = "Add New Subject"}
<<compositeUIElement>>
ShowCareer
{layout = BodyLayout}
<<list>>+name{listType = dropBox}
<<valueObject>>
CareerVO
+name
0..*
<<valueObject>>
SubjectVO
+code
+name
+semester
+hours
+program
<<compositeUIElement>>
SubjectsMng
{groupBy = "semester",
orderBy = "semester"}
<<plainText>>+semester{title = "Semester"}
<<plainText>>+code{title = "Code Subject"}
<<plainText>>+name{title = "Subject Name"}
<<plainText>>+hours{title = "Number of weekly hours"}
<<anchor>>+edit{title = "Edit Subject"}
<<anchor>>+delete{title = "Delete Subject"}

The *content diagram* allows modellers to specify the different elements that will be presented to final users in each page. The diagram consists of a set of presentation pages, each one related to a <<virtualState>> of the *node diagrams*, which contain one or more <<compositeUIElements>>. Each <<compositeUIElements>> class can have attributes classified as follows: static attributes, which represent static information not related to any other element of the different diagrams (e.g., the title of the web page or static text information); and binding attributes, which allows the transition from one state to another (e.g., a submit button). The presentation classes can also display information from a <<valueObject>> by establishing a dependency relationship between the class and a 'valueObject' defined in the logical layer diagram. Figure 15 shows the presentation page 'subject management' which is made up of two <<compositeUIElements>>: SubjectMng and ShowCareers. The composite element ShowCareers, contains a DropBox attribute to display all the available careers, and an association with the <<compositeUIElement>> SubjectsMng, to display all available subjects of a specific career. It is worth noting that the data that will be shown in the name attribute of ShowCareers, is defined by the dependency relationship between ShowCareers and CareersVO (this is also true for SubjectMng and SubjectVO). Finally, groupBy and orderBy tagged values defined for SubjectMng allows grouping and ordering subjects by semester.

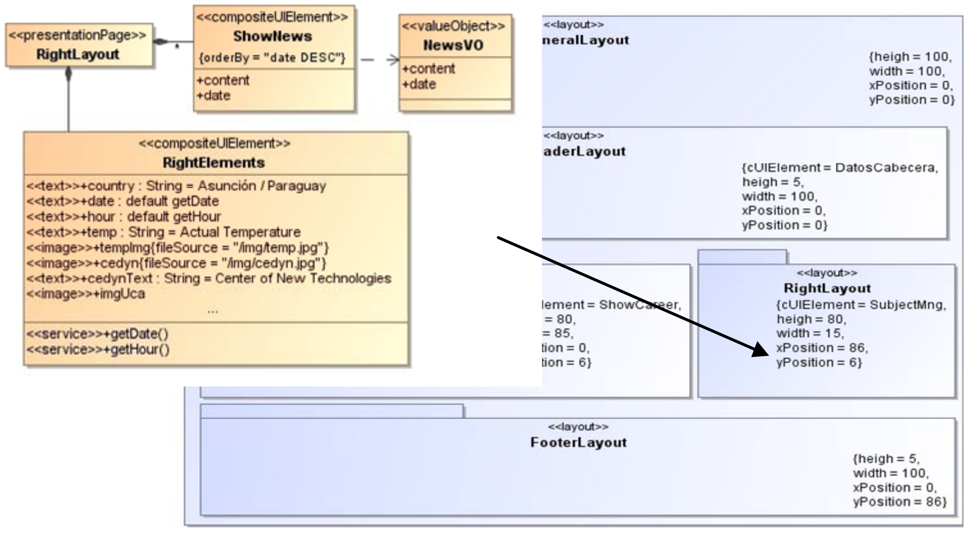
Figure 16 shows the *presentation diagram* composed of one or more *PresentationPages*, which aggregate different *PD elements*. The *PD elements* are classified into *UIElements* and *CompositeUIElements*. *UIElements* in turn are specialised into *anchor*, *TextInput*, *button*, *text*, *list*, *htmlText*, *Multimedia* and *ExternalLinks*. Each element has properties in order to model additional aspects related to constraints, limitations, possible values, among others.

Figure 16 Content metamodel and UML profile (see online version for colours)



The *structure diagram* is used for the definition of page areas (e.g., header, footer, or menu areas). UML packages stereotyped with <<layout>> represent regions. Each region can be composed of other sub-regions, and it is possible to define different layout structures for the same application (e.g., one structure diagram for each different target platform). It is also possible to define a basic content diagram for each region, which can then be complemented with the diagrams defined for each <<virtualState>>. An example of the latter is shown in Figure 15 and Figure 17. Figure 17 shows the basic content of the rightLayout region that will show the latest news available (ShowNews class), and some basic page information (RightElements class). On the other hand, Figure 15 shows the content diagram for the ‘subject management’ <<virtualState>>. This diagram indicates that the elements of the ‘SubjectMng’ class will be placed in ‘RigthLayout’ of the structure diagram, extending the basic content (news and basic information) of the region with the specific content of this page (SubjectMng elements). ShowCareers class, on the other side, will be placed in a different region of the structure diagram (‘BodyLayout’). Finally, to indicate the order in which presentation elements will be shown, a pair number property is defined, where the first number sets the vertical order and the second number the horizontal order.

Figure 17 Structure diagram and example of a content diagram for the ‘RighLayout’ (see online version for colours)



<<presentationPage>>
RightLayout
<<compositeUIElement>>
ShowNews
{orderBy = "date DESC"}
+content
+date
<<valueObject>>
NewsVO
+content
+date
<<compositeUIElement>>
RightElements
<<text>>+country : String = Asunción / Paraguay
<<text>>+date : default getDate
<<text>>+hour : default getHour
<<text>>+temp : String = Actual Temperature
<<image>>+tempImg{fileSource = "/img/temp.jpg"}
<<image>>+cedyn{fileSource = "/img/cedyn.jpg"}
<<text>>+cedynText : String = Center of New Technologies
<<image>>+imgUca
...
<<service>>+getDate()
<<service>>+getHour()
<<layout>>
neralLayout
{heigh = 100, width = 100, xPosition = 0, yPosition = 0}
<<layout>>
aderLayout
{cUIElement = DatosCabecera, heigh = 5, width = 100, xPosition = 0, yPosition = 0}
lement = ShowCareer, = 80, = 85, tion = 0, tion = 6}
<<layout>>
RightLayout
{cUIElement = SubjectMng, heigh = 80, width = 15, xPosition = 86, yPosition = 6}
<<layout>>
FooterLayout
{heigh = 5, width = 100, xPosition = 0, yPosition = 86}

With respect to the presentation style, even though it is considered a relevant aspect for the presentation layer, in our vision it is more reasonable to deal with style specifications in the ISM phase. Reasons for this decision are the style being very changing and normally taken into account in the final stages of development; the lack of a standard language at the modelling phase to specify this aspect and; the possibility to separately differentiate style from other aspects, allowing modifications of the application without changing any code (e.g., with CSS templates we could change the style at any time, affecting the appearance of the application).

Figure 18 Structure diagram metamodel and UML profile (see online version for colours)

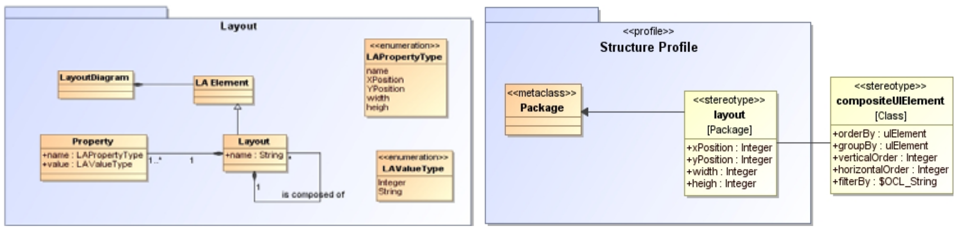
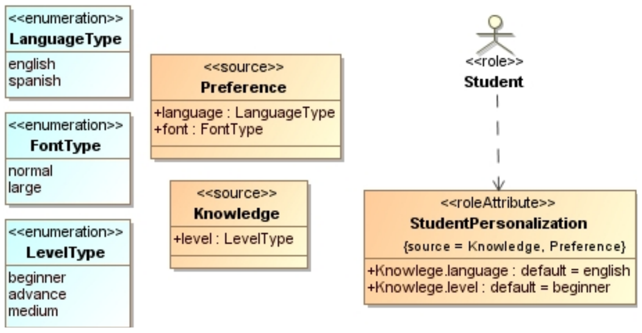


Figure 18 depicts the *structure diagram* metamodel, which is mainly composed of *LD elements*. The *LD elements* are classified into *layout*, which can be composed of other *layouts*. The *layouts* define dimensions and positions properties.

3.5 Stage 5: specify personalisation

According to Weibelzahl, personalisation refers to both adaptability and adaptivity (Weibelzahl, 2002). Adaptability requires user interaction in order to conceive personalisation (e.g., change colours, or types). On the other hand, adaptivity allows personalisation considering other factors without a direct user intervention (e.g., suggest list of books based on previous purchases). In order to consider these concepts, MoWebA defines two diagrams: *information source and rule diagrams*.

Figure 19 Source information diagram for web-based academic system (see online version for colours)



The *information source diagram* models user information needs for adaptation. The information sources refer to the system domain factors to be considered for rule conditioning, (e.g., in the example, an information source could be the level of knowledge for specific users). The next step is to define associations between sources and users considering the roles that they should play in the system. Therefore, we define a set of information sources and associate them with a given role; these are stereotyped with <<roleAttribute>>. The <<roleAttribute>> stereotype is used to establish relationships between sources and roles, and it is possible to set default values to these attributes. Figure 19 shows an example, we have defined two sources (preference and knowledge), assigned roleAttributes to the student, and assigned default values to these attributes (language = English and level = beginner). Such default values could be changed at any time in the future.

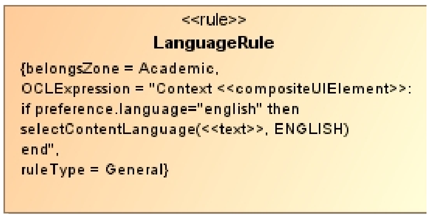
The *rules diagram* allows the definition of 'condition-action' rules that establish under which conditions a rule must be triggered in order to perform a specific action. The final result will be a dynamic adaptation of the system. An example of an adaptivity personalisation is a rule defined to filter exercise examples, the filtering could be done based on types of exercise that the student has already solved.

There are two types of rules:

1 general rules (e.g., if language is set to 'English', whenever a <<text>> element appears, it should be in English)

2 specific rules applied to specific elements (e.g., even though the font type is set to 'normal', a specific title of a page should be 'large').

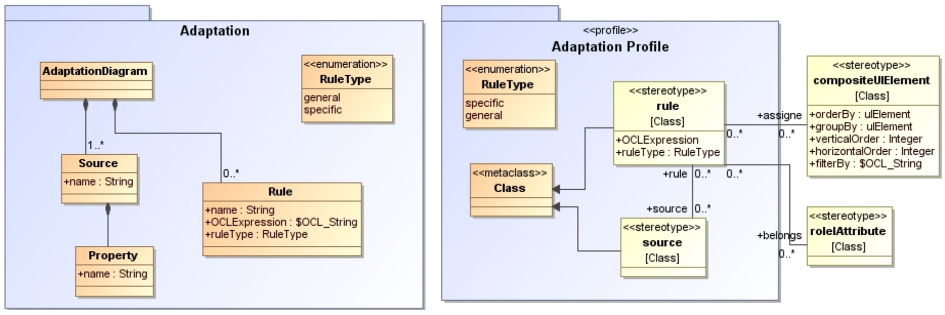
Figure 20 Rule example for language definition (see online version for colours)



```
<<rule>>
LanguageRule
{belongsZone = Academic,
OCLExpression = "Context <<compositeUIElement>>:
if preference.language="english" then
selectContentLanguage(<<text>>, ENGLISH)
end",
ruleType = General}
```

Rules are specified using an OCL expression as the tagged value of the class. For example, in Figure 20, the general rule called 'LanguageRule' has been defined for <<compositeUIElements>> of the content diagrams, belonging to academic zone (i.e., the zone associated to the student and professor roles). The OCL expression defines a condition related to the language attribute, triggering the *selectContentLanguage* action if the default language is 'English'. The behaviour of the *selectContentLangage* action must be specified in some way. In order to do this we define a process in the logic layer diagram called *AdaptationService*, and add the action *selectContentLanguage* as a <<service>> operation. The detailed behaviour of the selectContentLanguage <<service>> is then modelled in the service diagram, which will we be explained in the next section.

Figure 21 Adaptation metamodel and UML profile (see online version for colours)



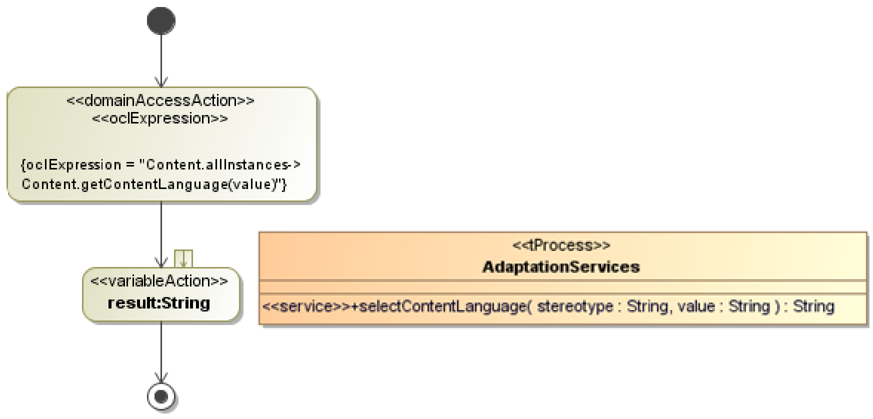
An *adaptation diagram* is composed of *rules* and *sources* (see Figure 21). For each *rule* we can specify a series of properties (*name*, *OCLExpression* and *rule type*). The *rules* can be associated to one or more *roleAttributes* of the *role diagram*, as well as one or more *compositeULElement* of the content diagram.

3.6 Stage 6: detail navigational, logic, adaptation and presentation services

Behavioural actions for each service specified at the navigational, logic, adaptation, and presentation diagrams can be modelled through the MoWebA *services diagrams*. The *service diagrams* use UML activity diagrams enriched with OCL and action semantics[1]. For each service/action defined in the other diagrams, it is possible to create a *service diagram* that encapsulates the associated service behaviour. Services are defined in the logic layer diagram and could be invoked by entities, rules, node or content diagrams elements.

To specify behavioural actions we use a set of basic and fundamental constructors. The basic constructors represent actions, transitions and pseudo-states. Fundamentals constructors consist of action specialisations classified into: *CallBehaviorAction*, representing a type of action that can invoke other behaviour; *DomainAccessAction*, representing access to the Entity model to perform an operation on it; and *VariableAction*, representing a special type of action whose implementation performs various operations on variables. Figure 22 shows the *service diagram* for the *selectContentLanguage* action, invoked by the rule 'languageRule' (see Figure 20).

Figure 22 Adaptation service (see online version for colours)



<<domainAccessAction>>
<<oclExpression>>
{oclExpression = "Content.allInstances->Content.getContentLanguage(value)"}
<<variableAction>>
result:String
<<tProcess>>
AdaptationServices
<<service>>+selectContentLanguage(stereotype : String, value : String) : String

Services allow the definition of behaviour actions at the modelling phase. In some situations a service diagram can be very complicated, because of the complex logic that it represents. In this case the service diagram definition could be avoided leaving the task of definition for the ISM phase.

The main idea of the service metamodel (see Figure 23) is to define specialisations of *Action*, which will enable to define more complex behaviours in the metamodel. The metaclass *CallBehaviorAction* represents a special kind of action that can invoke other behaviours represented by an activity diagram, or a behaviour that will come built into the final platform destination. In the figure, there are listed others specialisation of *action* (*variableAction*, *domainAccessAction* and *writePage*), and their relationships with other classes. The corresponding UML Profile for the Service metamodel is presented in Figure 24.

Figure 23 Service metamodel (see online version for colours)

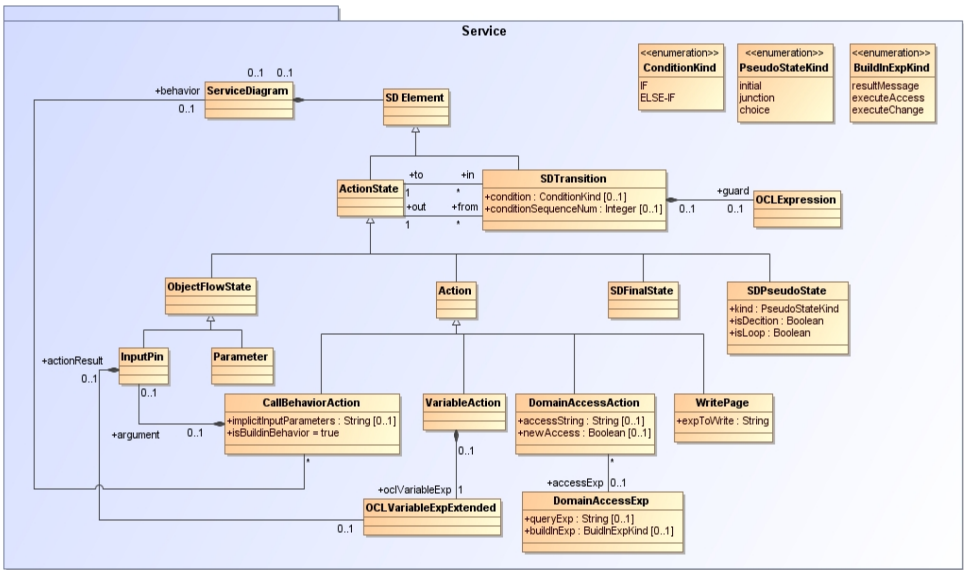
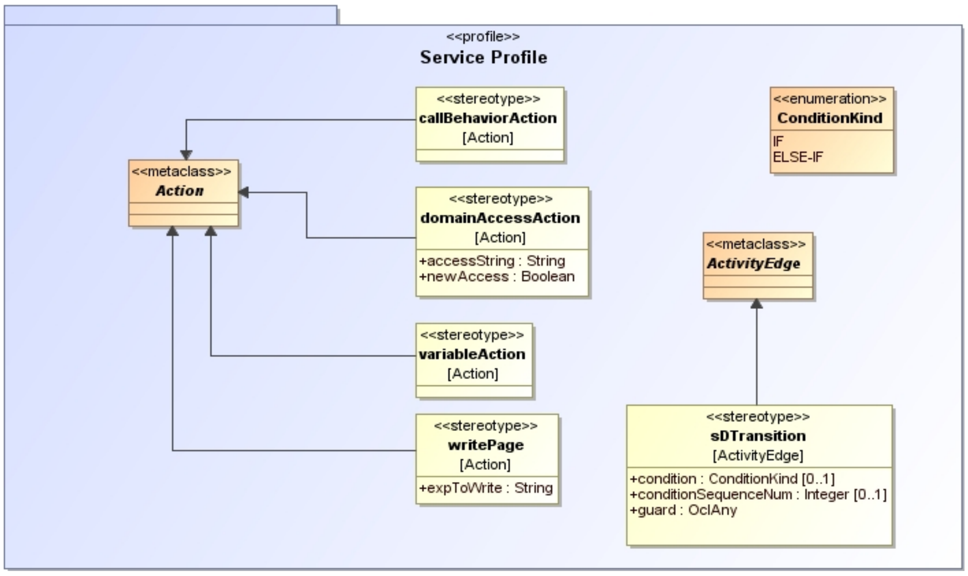


Figure 24 Service profile (see online version for colours)

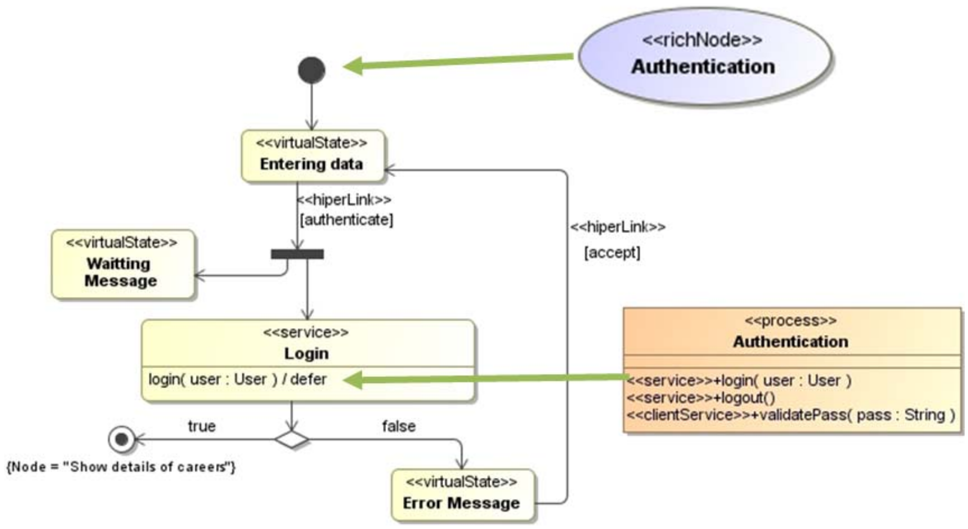

<<profile>>
Service Profile
<<stereotype>> callBehaviorAction [Action]
<<enumeration>> ConditionKind
IF
ELSE-IF
<<metaclass>> Action
<<stereotype>> domainAccessAction [Action]
+accessString : String
+newAccess : Boolean
<<metaclass>> ActivityEdge
<<stereotype>> variableAction [Action]
<<stereotype>> writePage [Action]
+expToWrite : String
<<stereotype>> sDTransition [ActivityEdge]
+condition : ConditionKind [0..1]
+conditionSequenceNum : Integer [0..1]
+guard : OclAny

3.7 Stage 7: ASM and PSM definition

Stage 7 is composed of two different models, which are generated in a semi-automatic way from the diagrams defined during the previous stages: the *architectural specific model* (ASM) and the *platform specific model* (PSM). ASM enriches the previous models with additional information related to the system architecture (e.g., RIAs, REST, among others). PSM is oriented to refine the models by adding information related to the platform and language selected for the final system (e.g., Java, .NET, PostgreSQL, among others). At this stage, we are moving from the conceptual definition (CIM/PIM models) to the solution definition (ASM/PSM models).

It is important to mention that other approaches generally include architectural aspects at the conceptual modelling level, without making a clear distinction between the independent model and the architectural one. For example, in order to generate RIAs, current approaches extend their notations with additional primitives or patterns considered at the conceptual modelling phase [e.g., WebML RIA (Fraternali et al., 2010), UWE for RIA (Koch et al., 2009)]. In MoWebA, the PIM could be used for different architectures (e.g., RIAs, REST, client-server, SOAs) since architectural aspects are not contemplated in this model. Therefore, MoWebA makes a clear separation between the conceptual space and architectural aspects, defining them on different modelling abstraction levels. In this way, our approach offers enough flexibility to evolve into different architectures starting from the same PIM model.

Figure 25 Navigational node applying the ASMRia model (see online version for colours)



The *ASM model* could be defined for the RIA architecture, obtaining an *ASMRia*. RIAs are web applications, which use data that can be processed both by the server and the client. The data exchange takes place in an asynchronous way, so that the client stays responsive while continuously recalculating or updating parts of the user interface. RIAs main characteristics are: data and page computation distribution, asynchronous communication between client and server, and enhanced user interface behaviour (Bozzon et al., 2006; Busch and Koch, 2009). In order to model these characteristics in an ASM model, MoWebA defines a series of stereotypes and tagged values. As an example of an *ASMRia* model for the academic system, Figure 25 shows the navigational node diagram for the 'authentication' node. The navigational node 'authentication' is stereotyped with <<richNode>>, meaning that everything inside this node will be executed mostly on the client side. Asynchronous communication is achieved for example by transitions modelled after the 'entering data' virtual state, since user validation is processed on the server. An example of a client side service could be 'validatePass' stereotyped with <<clientService>>. This service should be invoked at the presentation layer when the user sets a password in order to validate security levels.

Figure 26 ASMRia metamodel (see online version for colours)

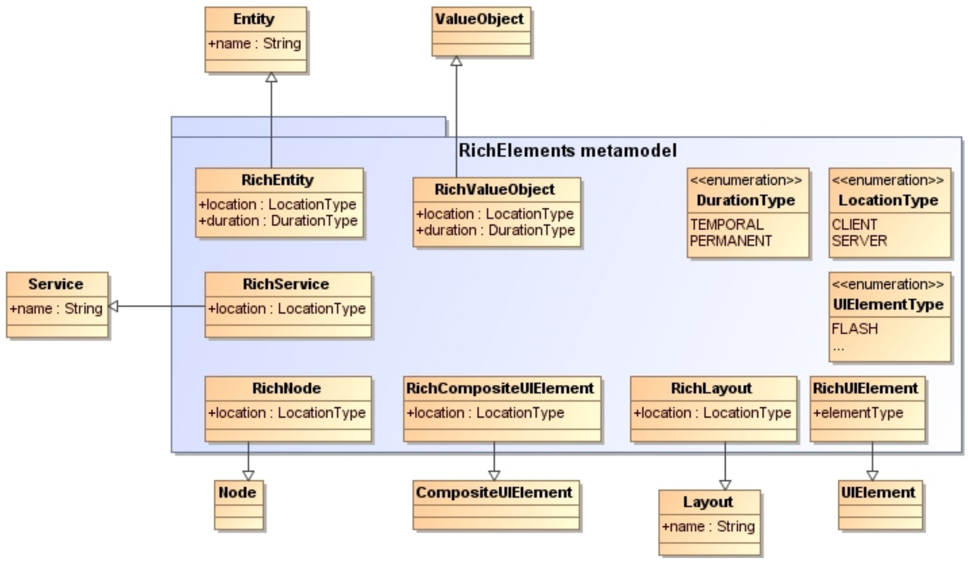


Figure 26 shows a first version of the *PSMRia* metamodel. In this metamodel, we show the extensions made on different elements related to distribution (client/server) and duration of persistent data and services. We are working on a more complete definition of an *ASMRia* considering presentation patterns, synchronisation, among other.

The *PSM model* enriches the models with specific platform information as the MDA approach suggests. In this sense, we can have one or more PSM models depending on the target platform selected for the application. In the example, one of the target platforms is Ruby on Rails. For this purpose, we have defined a *PSMRuby metamodel* presented in Figure 28. In this figure it is possible to notice that presentation elements are redefined according to Ruby on Rails platform.

Figure 27 Content diagram for the 'entering data' virtual state (see online version for colours)

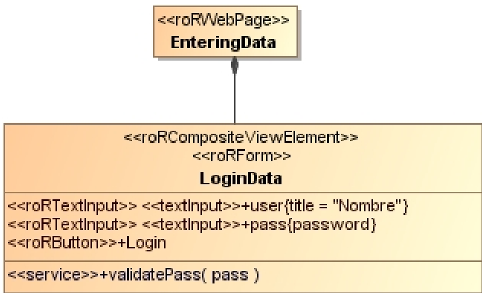
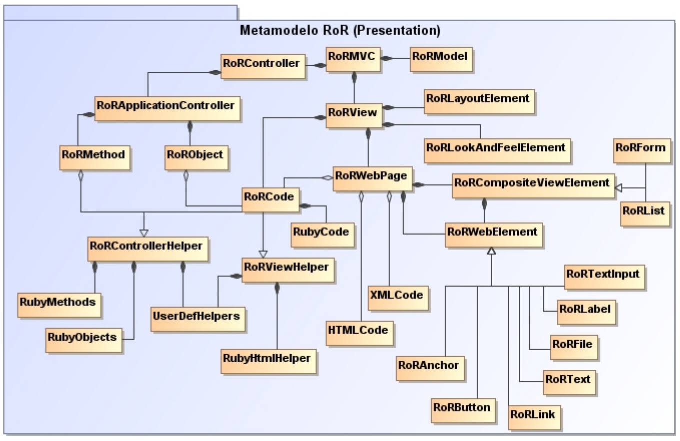

<<roRWebPage>>
EnteringData
<<roRCompositeViewElement>>
<<roRForm>>
LoginData
<<roRTextInput>> <<textInput>>+user{title = "Nombre"}
<<roRTextInput>> <<textInput>>+pass{password}
<<roRButton>>+Login
<<service>>+validatePass(pass)

Figure 27 presents a content diagram with a *PSMRuby* extension for a Ruby on Rails platform.

The ASM and the PSM can be defined and included into the model as plug-in extensions. Indeed, to consider emerging web technologies, MoWebA proposes to define a new ASM and/or PSM metamodel.

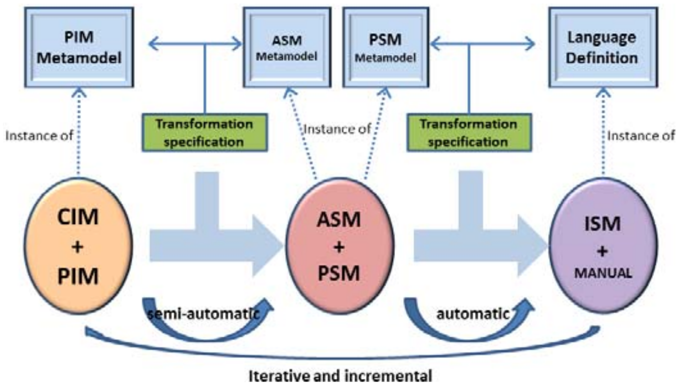
Figure 28 PSM ruby metamodel (see online version for colours)



4 MoWebA transformation process

The transformation process implies steps and activities for transformation specification in order to go through each MoWebA phase (CIM/PIM-ASM/PSM, ASM/PSM-ISM/Manual). This process aims to define intermediate specific models before the final implementation (see Figure 29).

Figure 29 MoWebA transformation process (see online version for colours)



The transformation process is based on metamodels (PIM-ASM-PSM transformation). The PIM-ASM/PSM phase is done in a semi-automatic way; since sometimes the information to be added requires human intervention (e.g., in RIAs, the modeller needs to specify where services will be executed, on the client or on the server). The automation of this process is done using a MDD standard such as QVT, along with a tool that supports this standard (e.g., operational QVT). An example of the QVT transformation rule is shown in Figure 30. In this figure, the QVT transformation rule is defined by using the relation language, in order to transform the MoWebA *entity diagram* (which corresponds to the input model) in a *PSMPostgres* (which corresponds to the output model) diagram. Input and output diagrams vary according to each specific QVT transformation rule.

Figure 30 QVT definition to obtain the PSMPostgres diagram

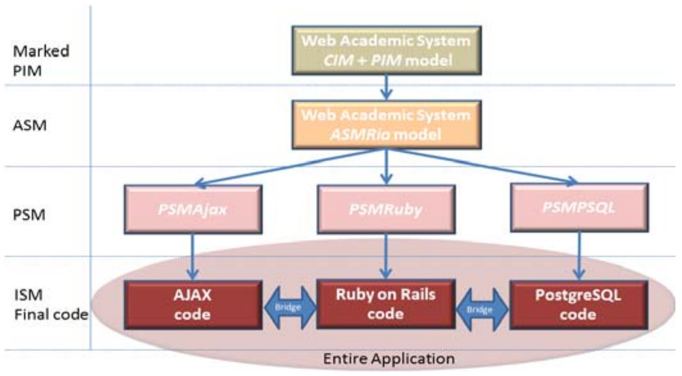
```
top relation EntityToTable {
    prefix, eName: String;

    checkonly domain entityDiagram
    entity:Entity {
        name = eName
    };

    enforce domain PSMPostgres table:Table
    {
        name = eName
    };
    where {
        prefix = ‘‘;
        RecordToColumns(entity, table,
        prefix);
    }
```

```
relation RecordToColumns {
    checkonly domain entityDiagram
    record:Record {
        fields = field:Field { }

    };
    enforce domain PSMPostgres table:Table
    { };
    primitive domain prefix:String;

    where {
        FieldToColumns(field, table);
    }
}
```

The ASM/PSM-ISM phase is done automatically by using open source tools (e.g., Acceleo, AndroMDA). The input models of this phase are the *PSMs* obtained at the previous phase, and the output will be se *source code*.

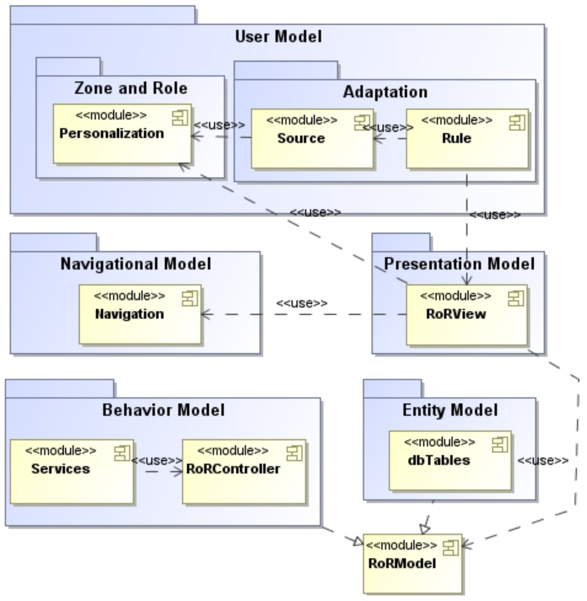
We refer to the final implementation of the System as *ISM*. The ISM will contain code for every platform selected and the bridges between them, in order to get a functional system ready to be deployed. We have experienced two types of ISM obtained by defining transformation rules with two different tools: AndroMDA and Acceleo.

Figure 31 The web-based system transformation process (see online version for colours)



In order to implement the MoWebA transformation rules, we defined a series of modules (shown in Figure 32). For reasons of space, we will only explain in detail the source and rule models, defined for the adaptation code generation phase.

Figure 32 Acceleo modules definition for MoWebA (see online version for colours)



The transformation process for our web academic system example is shown in Figure 31.

The academic system was generated using the Acceleo Tool. Acceleo is considered a template-based M2T (model to text) transformation open source MDD tool, which adopts the model to text language (MTL) standard for transformation rules definition[2]. This tool was created in 2006 as a part of the eclipse modelling project (EMP)[3]. The Acceleo code generation process considers the following steps:

1 code generator project creation

2 input models inclusion (XMI files)

3 modules definition and templates creation

4 associated services creation

5 code generation

6 project depuration

7 generators modules exportation.

Modules are considered as partial or full implementations of transformation rules for a specific platform. They can be executed as plug-ins of eclipse to generate an application in the target platform. Modules are composed of templates, services and queries written in the Java programming language. Templates use a specific syntax composed of tags. Queries are used to extract information from the model, which can return values or collections. Java services are used to define complex or common operations that can be accessed by the different templates defined within the module.

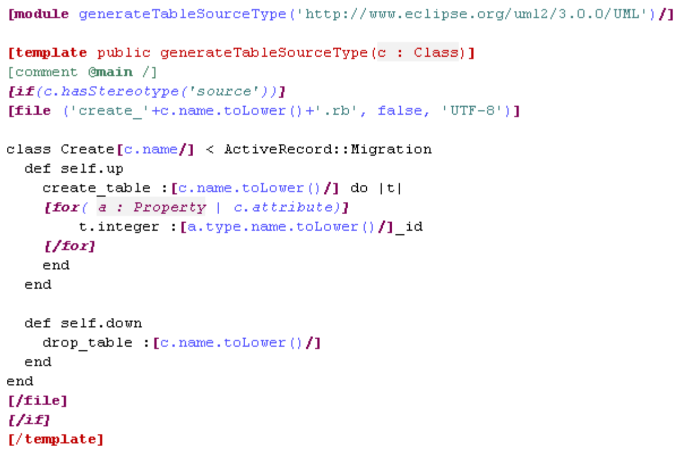
The adaptation transformation rules are composed of the source and rule modules. The source module contains templates defined for information source generation and the rule module corresponds to the adaptation rules processing.

The source module is composed of the following templates:

- generateTableSource: creates the database tables with the parameters defined in the information source model
- loadSources: generates a file with SQL sentences to insert possible values defined in enumerations
- generateTableSourceType: generates ruby files for modules in order to manipulate the database tables
- generateSourcesForRoleAttribute: associates a user with a specific role, and information sources with default values defined in the model.

Figure 33 shows the generateTableSourceType template.

Figure 33 generateTableSourceType.mtl template (see online version for colours)



```
[module generateTableSourceType('http://www.eclipse.org/uml2/3.0.0/UML')/]

[template public generateTableSourceType(c : Class)]
[comment @main /]
[if(c.hasStereotype('source'))]
[file ('create_'+c.name.toLower()+'.rb', false, 'UTF-8')]

class Create[c.name/] < ActiveRecord::Migration
  def self.up
    create_table :[c.name.toLower()/] do |t|
    [for( a : Property | c.attribute)]
        t.integer :[a.type.name.toLower()/]_id
    [/for]
    end
  end

  def self.down
    drop_table :[c.name.toLower()/]
  end
end
[/file]
[/if]
[/template]
```

The rule model, on the other side is composed of:

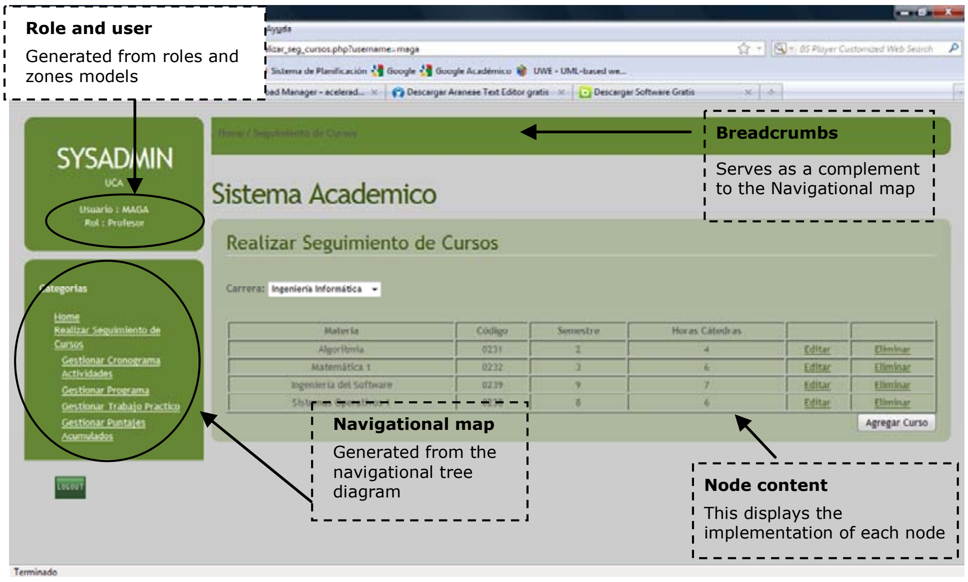
- generalRuleTransformation: is applied to the rule classes stereotyped with <<rule>> and isGeneral=True. This template is composed of auxiliary templates: getOclExpression, to retrieve the OCL expression; getSource, to identify the source referencing; and sourceType, to identify the source type.
- applyGeneralRule.mtl: is defined to apply the general rule to the presentation elements.
- specificRuleTransformation.mtl: analyses the specific rules, retrieving the OCL expressions, sources and actions.
- applySpecificRule.mtl: applies the specific rule to the presentation elements associated to it.

Figure 34 shows the generalRuleTransoformation.mtl template.

Figure 34 generateRuleTransformation.mtl template (see online version for colours)

```
[comment encoding = UTF-8 /]
[module generalRuleTransformation('http://www.eclipse.org/uml2/3.0.0/UML')/]

[template public generalRuleTransformation(c : Class) post (trim())]
[comment @main /]
[if (c.hasStereotype('rule'))]
[for (a : Stereotype | c.getAppliedStereotypes())]
    [if (c.isGeneral(c.getValue(a, 'ruleType').toString()))]
        [file (c.name, false, 'UTF-8')]
def self.get_[c.getSource(a)/]_[c.sourceType(a)/]([c.getSource(a)/]_id)
    if [c.getSource(a)/]_id
        @[c.sourceType(a)/] = [c.sourceType(a).toString().toUpperFirst()/].find_by_id(
        [c.getSource(a).toString().toUpperFirst()/].find_by_id([c.getSource(a)/]_id).
        [c.sourceType(a)/]_id)
    end
    return @[c.sourceType(a)/]
end
        [/file]
    [/if]
[/for]
[/if]
[/template]
```

Figure 35 An example of a generated page (see online version for colours)


Role and user
Generated from roles and zones models
SYSADMIN
UCA
Usuario : MAGA
Rol : Profesor
Categorias
Home
Realizar Seguimiento de Cursos
Gestionar Cronograma Actividades
Gestionar Programa
Gestionar Trabajo Practico
Gestionar Puntajes Acumulados
LOGOUT
Breadcrumbs
Serves as a complement to the Navigational map
Sistema Academico
Realizar Seguimiento de Cursos
Carrera: Ingeniería Informática
Agregar Curso
Navigational map
Generated from the navigational tree diagram
Node content
This displays the implementation of each node
Terminado

Materia	Código	Semestre	Horas Cátedras		
Algoritmia	0231	2	4	Editar	Eliminar
Matemática 1	0232	3	6	Editar	Eliminar
Ingeniería del Software	0239	9	7	Editar	Eliminar
Sistemas Operativos 1	[illegible]	8	6	Editar	Eliminar

Figure 35 shows an example of a page of the web academic system resulting from the transformation process. In this figure we can visualise some parts generated from the MoWebA models (e.g., from the navigational tree, node content, and roles and zones diagrams).

5 Adopting MoWebA: some experiences

MoWebA has been used for modelling and generating different types of applications by novice and experienced modellers and developers. Experienced modellers were already familiar the UML notation and web methodologies (e.g., UWE, WebML, OOWS, or OOHDM), while developers were experienced with different programing languages.

Table 1 Experiences with MoWebA

Application	*Type*	*Aspects considered*			
		Team[a]	*Profiles*	*Type of project*	*Analysis*
Online course	e-learning	2EM	Professionals	Academic	Interview
		1ED	Thesis students		
University administration	Administration	12NM	Students	Academic	Interview
Aquatic birds portal	Management	4EM	Professionals	Real project	Interview
Academic system	e-learning	2 EM	Professionals	Academic	Interview
		2 ED			
Laboratory management	Management	3 EM	Thesis students	Academic	Questionnaire
		3D	Thesis students		
		12 NM	Students		
Budget execution	Administration	4 NM	Students	Real project	Interview
		4 MD	Advance students		
Surveys	Interactive	3 NM	Students	Real project	Interview
Social network	Community	12 MM	Advance students	Academic	Interview
		12 MD	Advance students		

Notes: [a]Team: level E (experienced), N (new), M (medium); type M (modeller), D (developer).

These experiences, which are summarised in Table 1, are proofs of concepts in academic and industrial settings. They have offered insights for improving specific aspects of the processes and of different models of MoWebA. In addition, they are paving the way for a more rigorous validation of the proposal in which we are currently working. The experiences relied on two types of validation instruments (i.e., interviews and questionnaires) in order to identify strengths and weaknesses.

Table 2 Aspects of MoWebA adoption in the different experiences

Application	*Modelling aspects analysed*					*Development aspect considered*		
	UC	*Nodes*	*Classes*	*Pres. pages*	*Services*	*Develop time*	*Target platform*	*Tool adopted*
Online course	32	28	23	59	48			
University admin	98	92	72	247	248			
Aquatic birds portal	95	109	25	266	83			
Academic system	20	35	22	105	93	Six months	Ruby on Rails	Acceleo
Lab management	15	17	13	28	19	Four months	PHP	AndroMDA
Budget execution	27	19	16	79	26	Six months	PHP-Zend	Acceleo
Surveys	12	21	14	35	25	Six months	PHP-Zend	Acceleo
Social network	17	38	12	40	26	Fou months	Ruby on Rails	Acceleo

For a more objective analysis, Table 2 summarises the diverse characteristics of these applications. Some characteristics are related to the complexity of applications and modelling elements, and others to the development process. A summary of the most important considerations arising from these experiences are presented below:

- A first positive aspect is that Navigational structures considered were easy to model, and easy to understand by subjects. For example, the academic system is composed of 35 navigational nodes, with a mean of three virtual states per node, where each virtual state represents a page. Having a global hierarchical view of the system with 35 elements is more manageable than 105 pages.
- The node diagrams were helpful to identify behavioural and presentation elements more easily. We could note that for each navigational node there were identified, in average, two to three services and three to four virtual pages. Thus, it is possible to decompose the overall navigational structure into smaller parts, taking into account the specific behavioural navigation for each functional element.
- The CIM/PIM phase was standardised, and could be modelled with any tool that supports UML 2.0 (e.g., Magic Draw and Papyrus). The generated models were exported to the XMI format in order to integrate them with Acceleo and AndroMDA. Even though it was possible to work with different tools, some details had to be considered, especially specially when defining tagged values.
- The automation was performed using two different tools: AndroMDA and Acceleo. On average, the automatic generation percentages for each layer were the following: data layer, 100%; logic layer, 61%; navigational layer, 100%; and presentation layer, 73%. The reason for logic layer not being totally generated is that some services were difficult to model because of their behavioural complexity; therefore they had to be added manually. With respect to presentation, there are some aspects related to style (e.g., fonts, colours, among others) that can only be defined manually.
- MoWebA allows the modelling of diverse types of web applications. Even though, special characteristics e.g., such as RIAs or REST, need further specification For this reason, in order to add RIA characteristics to our web academic system example, we had to define the *ASMRia* model.
- One of the limitations we encountered was that services were sometimes difficult to model, but despite services not being totally defined, the PIM could be defined almost completely. We noticed that for service definition it was necessary to have knowledge in action semantics and OCL, but most of the modellers were not as experienced with these, as they are with UML. However, considering all the services defined in models for the different applications we saw that only 8.6% of the services were complex, while most of them were medium (30.5%) or simple (60.9%) services.

Furthermore, the transformation rules defined using AndroMDA and Acceleo, made it possible to generate code for three different target platforms: PHP, Python and Ruby on Rails.

In addition to the experiences with different types of applications, some modelling experiences with different user profiles (i.e., expert, and novice modellers) were also carried out. An interesting experience was carried out with two groups, one formed entirely by students and the other entirely by MDD experts, for analysing the quality of the MoWebA models. It was focused on studying different perspectives such as simplicity, abstraction, ease of use, among others, adopting the GQM approach (Basili et al., 1994). The results were quite positive, although there were some difficulties with the modelling, especially for the service diagrams, because of their complexity.

6 Related works and discussions on MoWebA

Schwinger and Koch classified the web methods following different paradigms (Schwinger and Koch, 2006): data-oriented, hypertext-oriented, object-oriented and software-oriented. In this work, we present an improvement over the web method classification table proposed by Schwinger and Koch. First, by considering new trends in methodologies, we propose the MDD-oriented paradigm as a new category. Second, we believe that the paradigms are not totally independent and therefore a methodology could be classified in more than one paradigm. Third, we eliminate some methodologies that are no longer in use.

Table 3 Web method classification

Method	*Classification*				
	Hypertext-oriented	*Data-oriented*	*Object-oriented*	*Software-oriented*	*MDD-oriented*
W2000 (Baresi et al., 2006)	X		X		
Hera (Houben et al., 2004)		X			
WebML (Ceri et al., 2000)		X			X
WSDM (De Troyer and Decruyenaere, 1998)	X			X	
OOHDM (Schwabe and Rossi, 1998)		X	X	X	
UWE (Koch et al., 2007)		X	X		X
OO-H (Cachero et al., 2000)		X	X		X
OOWS (Fons et al., 2007)		X	X		X
WAE2 (Conallen, 2003)					X
WebSA (Meliá et al., 2005)			X		X
MoWebA	X		X		X
UWA (Distante et al., 2007; Bernardi et al., 2014)	X		X		X

Next, we present a discussion of MoWebA for each concern in comparison with related works.

C1 Navigational oriented modelling could help to simplify the models for web applications

In MoWebA the navigational model is the central and starting point for modelling (see stage 2). This approach is an alternative way to model the navigational perspective better fitting the requirements of users' interaction and making user navigation more adherent to its mental model. From the first experiences, it seems to support modellers in defining functions oriented to navigational structures, thus simplifying user orientation.

Other similar interesting approaches are: UWA (based on W2000) and WSDM. However, in UWA navigation and services are derived from the information model. Despite deriving the navigational model from the structural model may be useful in order to organise the information content, it does not model users' interaction in all their dimensions, as already presented in the introduction section. The WSDM proposal regarding the navigational structure (i.e., function oriented) is quite similar to the MoWebA approach. However, some differences are:

1 the navigational model is more complex (e.g., including a great amount of constructors)

2 the method does not use a standard notation and it does not support automation tools (refers to the second concern); and, consequently

3 the evolution of technologies is difficult to manage (refers to the third concern).

MoWebA, as other approaches like OOHDM and OOWS, also discriminates between intra (e.g., hard and soft links) and inter-contextual (e.g., hyperlinks between states of the node diagram) navigations. However, it proposes their definition in a different way:

1 the concept underlying the soft links (i.e., navigate to an unrelated node in the navigational tree) is quite singular of the MoWebA approach

2 the incremental process definition starts with the identification of the hard links in the navigational tree diagram as a first step, and the hyperlinks and softLinks of the node diagram as a second step.

This simplifies the overall understanding of the application structure and makes a distinction between the different levels of navigation (see the first consideration in Section 5).

C2 The adoption of standards will facilitate the interoperability between models, methods, and transformation rules.

MoWebA, like other methodologies to some extent, adopts the MDD standards and follows the object-oriented paradigm in every phase (e.g., UWE, OO-H, and OOWS). MoWebA works with different tools and notations:

1 it formalises the processes by applying the MOF metamodelling language

2 it adopts UML profiles in XMI format to allow modelling with different case tools

3 it defines model to model transformation rules for PIM-ASM/PSM, adopting QVT where the proposed ASM is RIA, and the proposed PSMs are PHP, Ruby on Rails and PostgreSQL

4 it defines model to text (i.e., code generation) rules using the Java and ATL language with the support of two different tools (AndroMDA and Acceleo).

MoWebA adopts the MDA approach in all the standards and in the entirely process trying to make profit from all the MDD potential for web engineering.

In our best knowledge, UWE is the only other methodology whose models and processes completely follows the MDA approach, maintaining some differences regards the other concerns. Moreover, another difference resides in the generation process. UWE code generation process is done in a semi-automatic way, since the generated code requires additional adjustments for obtaining the final application (e.g., UWE4JSF which works in the eclipse environment and generates JSF applications requiring additional adjustments for some java classes, libraries, stylesheets, among others). MoWebA also follows the semi-automation approach, but this is done in the PIM-ASM/PSM phase, since human intervention is needed to decide some transformation rules (see Section 4).

C3 Take into account the evolution of web environments for improving the development of current web applications.

MoWebA considers evolution in different aspects. At a more structural level, considering the evolution of the architecture and the final implementing platform, to the best of our knowledge an approach such as the one from MoWebA has not been presented by any other methodology. MoWebA separates the PIM, ASM and PSM models, in order to facilitate the evolution of applications. With this separation, a clear distinction is made between what would be the problem space, presenting a model that is completely independent of the target architecture or platform; and the solution space, through the ASM, PSM and the final code. Such proposal, tend to facilitate the support of web development for the Web 2.0. In fact, for achieving dynamic website, where the users actively interact with the web application, it is common to use technologies and platforms like web services and RIAs, among others. Methodologies such as WebML and UWE propose extensions for RIAs (or other final platforms) during the PIM modelling phase. However, such extensions reduce the reusability and portability of the PIM. By contrast, MoWebA captures the requirements for specific platforms at the ASM model according a semi-automated process. As a counterpart, to offer a greater reusability of the PIM facilitating the architectural evolution of the web applications, the MoWebA approach requires some additional effort, including the need for metamodels and the definition of the corresponding transformation rules, to achieve automatic transformations on the proposed architecture or platform.

Moreover, the evolution concern is not only one for architectural/technological issues, since the functional requirements of web applications also evolve fast. MoWebA, as well the other methodologies that adhere to the MDD approach, follows an incremental process, facilitating such type of functional changes that are defined at the model level which will then be transformed into code by using automated tools.

At a user adaptation level, in general, to cover the accessibility aspects, other methodologies propose notations to model user groups (e.g., OOHDM and OO-H). These are defined in MoWebA with the *zone diagram*, which also allows setting different user levels (groups of users that are defined by the roles, the roles related to each other through the zones and zones of the different levels that may arise). It is also possible to define access privileges on different notational elements, identifying different levels of security. Adaptation is also considered in other methodologies to allow personalisation strategies. For example, UWE defines adaptation using the aspect paradigm. With the MoWebA adaptation model it is possible to cover adaptability and adaptivity, with the *source information and rule diagrams*. In addition, in the presentation modelling stage of MoWebA (see Section 3.4), the separation between content and structure, allows more adaptability. In most methodologies there is no such distinction.

Finally, the presentation dimension is a critical aspect in web engineering and still requires more effort to assure an adequate automatic generation. Some proposals consider the separation of presentation and application logic to be necessary (e.g., UWE). Other proposals indicate the importance of establishing a clear separation between application, presentation and control logic, especially when multiple presentation channels should be served by the same application logic (Book and Gruhn, 2009; Horrocks, 1998). MoWebA considers these aspects with the logic (through value objects and services) and node diagrams (through virtual and service states). Nevertheless, future works are needed to deal with these open issues.

7 Conclusions and future work

This study presented MoWebA, a proposal for the development of web applications. MoWebA defines navigation from a behavioural point of view, instead of a structural (data-oriented) one, trying to better capturing the requirements of users' interaction, and it considers navigation as the starting point of the modelling process for web applications. Moreover, it includes an appropriate syntax to model the dynamic navigation observed during the users' interaction and the inter-intra contextual navigation. Another innovative contribution in MoWebA is the ASM – architectural specific model, which define an architectural level of modelling definition separated from the PIM, in order to facilitate the evolution of applications. MoWebA strongly adopts the standards proposed by MDD (languages, tools, architecture, among others) in every phase. An important effort is devoted to personalisation aspects. Based on the results of the various experiences performed, this study discusses the current proposals, highlighting the contributions and weaknesses of MoWebA in each phase.

Results were quite encouraging, and stimulated new ongoing experiences, case studies and more rigorous experiments: application of MoWebA and other methodologies (UWE, OOHDM, OO-H, WebML) to the same real applications; integration and refinement of rules definition in Acceleo; rules definition for other platforms (for example, J2EE); PSMs definition for different platforms; and validation of models.

Finally, we also consider that MoWebA has sufficient flexibility to support innovative technologies, such as those typical of Web 2.0 (e.g., *ASMRia* extension). To validate these considerations, proofs of concept and case studies that focus on building applications with current technologies (e.g., RIAs, REST, cloud computing) that facilitate development of Web 2.0 applications are being planned. We are also considering an interesting future work, to compare the development time required using MoWebA with regards to:

1 competing approaches

2 manually creating all the web apps without using models; considering updates activities for evolution analysis.

Acknowledgements

This research is partially supported by the CONACYT (Paraguay), through the INV-056 Project.

References

Acerbis, R., Bongio, A., Brambilla, M., Tisi, M., Ceri, S. and Tosetti, E. (2007) 'Developing eBusiness solutions with a model driven approach: the case of acer EMEA', *Proceedings of the 7th International Conference Web Engineering (ICWE'07)*, Berlin.

Baresi, L., Colazzo, S., Mainetti, L. and Morasca, S. (2006) 'W2000: a modelling notation for complex web applications', in *Web Engineering*, pp.335–364, Springer, Berling.

Basili, V., Caldiera, G. and Rombach, H. (1994) 'Goal question metric approach', in *Encyclopedia of Software Engineering*, pp.528–532, John Wiley & Sons, New York.

Bernardi, M., Di Lucca, G. and Distante, D. (2014) 'Model-driven fast prototyping of RIAs: from conceptual models to running applications', *3rd International Conference on Advances in Computing, Communications & Informatics, ICACCI 2015*, Delhi, India.

Blechar, M. and Norton, D. (2009) *Trends in Model-Driven Development*, p.3, Gartner Research, Technical Report, 4Q09-3Q10, ID Number: G00169442.

Book, M. and Gruhn, V. (2009) 'Fine-grained specification an control of data flows in web-based user interfaces', *Journal of Web Engineering*, Vol. 8, No. 1, pp.48–70.

Bozzon, A., Comai, S., Fraternali, P. and Toffetti, G. (2006) 'Capturing RIA concepts in a web modeling language', *15th International Conference on World Wide Web*, New York, USA.

Brambilla, M., Cabot, J. and Wimmer, M. (2012) *Model-Driven Software Engineering in Practice*, Morgan&Claypool, USA.

Busch, M. and Koch, N. (2009) *Rich Internet Application. State of the Art*, Technical Report 0902. Programming Software Engineering Unit (PST), Munchen, Germany.

Cachero, C. and Koch, N. (2002) 'Conceptual navigation analysis: a device and platform independent navigation specification', *2nd Internationa Workshop on Web-oriented Software Technology – IWWOST*, Málaga, Spain.

Cachero, C., Gómez, J. and Pastor, O. (2000) 'OO-HMethod: Un Método de Diseño de Lugares web', *IDEAS 2000*, Cancún.

Ceri, S., Fraternalli, P. and Bongio, A. (2000) 'Web modelling language: a modelling language for designing web sites', *Computer Networks*, Vol. 33, No. 1, pp.137–157.

Conallen, J. (2003) *Building Web Application with UML*, 2nd ed., Addison-Wesley, Massachutsets, USA.

De Troyer, O. and Casteleyn, S. (2003) 'Exploiting link types during the conceptual design of web sites', *International Journal of Web Engineering Technology*, Vol. 1, No. 1, pp.17–40.

De Troyer, O. and Decruyenaere, T. (1998) 'Conceptual modelling of web sites for end-users', *World Wide Web*, Vol. 3, No. 1, pp.27–42.

Deshpande, Y., Murugesan, S., Ginige, A., Hansen, S., Schwabe, D., Gaedke, M. and White, B. (2002) 'Web engineering', *Journal of Web Engineering*, Vol. 1, No. 1, pp.3–17.

Distante, D., Pedone, P., Rossi, G. and Canfora, G. (2007) 'Model-driven development of web applications with UWA, MVC and JavaServer faces', in *Web Engineerign*, pp.457–476, Springer Berlin Heidelberg, Como.

Fons, J., Pelechano, V., Pastor, O., Valderas, P. and Torres, V. (2007) 'Applying the OOWS model-driven approach for developing web applications: the internet movie database (IMDB) case study', in *Web Engineering: Modeling and Implementing Web Applications*, pp.65–108, Springer, London.

Fraternali, P., Comai, S., Bozzon, A. and Toffett, G. (2010) 'Engineering rich internet applications with a model-driven approach', *ACM Trans. Web*, Vol. 4, No. 2, p.7.

Gómez, J., Bia, A. and Parraga, A. (2005) 'Tool support for model-driven development of web applications', *Web Information System Engineering – WISE*, Berlin.

Horrocks, I. (1998) *Constructing the User Interface with Statecharts*, Addison-Wesley Professional, Bellingham, USA.

Houben, G., Frasincar, F., Barna, P. and Vdovjak, R. (2004) ‘Modeling user input and hypermedia dynamics in hera’, *4th Internationa Conference on Web Engineering*, Munich.

Koch, N., Knapp, A., Zhang, G. and Baumeiter, H. (2007) ‘UML-based web entineering, an approach based on standards’, in *Web Engineering: Modeling and Implementing Web Applications*, pp.157–192, London, Springer.

Koch, N., Pigerl, M., Zhang, G. and Morozova, T. (2009) ‘Patterns for the model-based development of RIAs’, *International Conference on Web Engineering (ICWE)*, San Sebastian, Spain.

MDA Guide Version 1.0.1 [online] http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf (accessed 5 July 2015).

Meliá, S., Gómez, J. and Koch, N. (2005) ‘Improving web design methods with architecture modeling’, *6th International Conference on Electronic Commerce and Web Technologies – EC-Web*, Copenhagen.

Mernik, M., Heering, J. and Sloane, A. (2005) ‘When and how to develop domain-specific languages’, *ACM Computing Surveys (CSUR)*, Vol. 37, No. 4, pp.316–344, DOI 10.1145/1118890.1118892.

Nuseibeh, B. and Easterbrook, S. (2000) ‘Requirements engineering: a roadmap’, *22nd International Conference on Software Engineering – ICSE*, Limerick Ireland.

Pressman, R. and Lowe, D. (2009) *Web Engineering: A Practitioner’s Approach*, McGraw-Hill, New York.

Rossi, G., Pastor, O., Schwabe, D. and Olsina, L. (2007) *Web Engineering: Modelling and Implementing Web Applications*, Springer, London.

Schwabe, D. and Rossi, G. (1998) ‘An object oriented approach to web-based application design’, *Theory and Practice of Object Systems*, Vol. 4, No. 4, pp.207–225.

Schwinger, W. and Koch, N. (2006) ‘Modeling web applications’, *Web Engineering: A New Discipline for Development of Web-Based Systems*, pp.39–64, John Wiley, New York.

Weibelzahl, S. (2002) *Evaluation of Adaptive Systems*, Dissertation Presented to the Faculty I of the University of Trier, Trier, Germany.

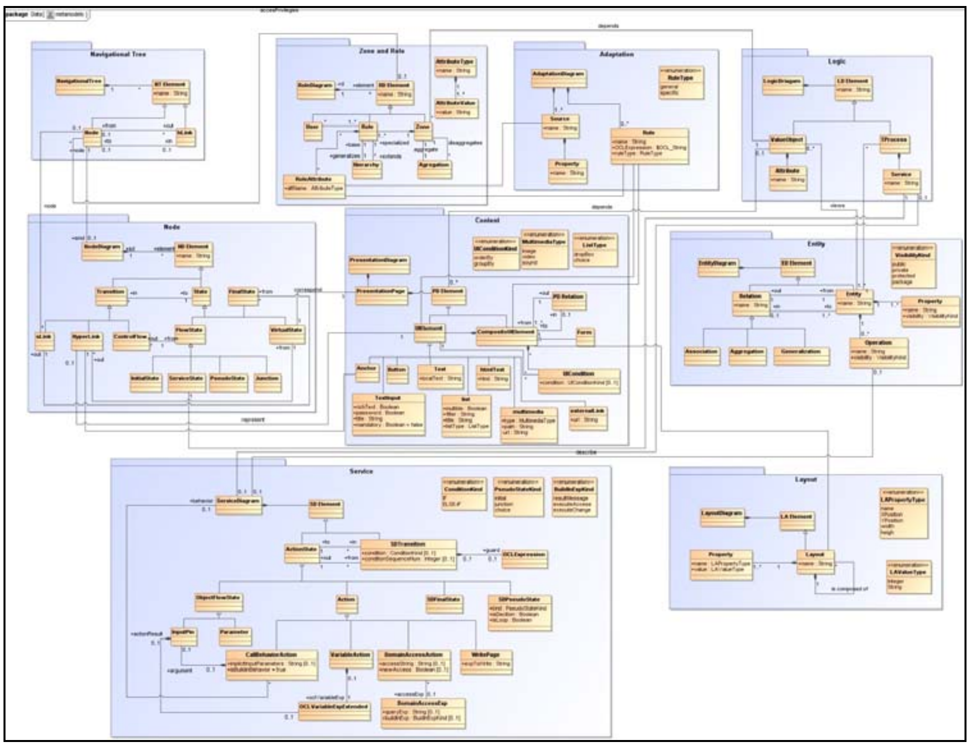
Notes

1 http://www.omg.org/docs/ptc/02-01-09.pdf.

2 http://www.omg.org/spec/MOFM2T/1.0/PDF.

3 http://www.eclipse.org/modeling/.

Appendix (see online version for colours)


Navigational Tree
Zone and Role
Adaptation
Logic
Node
Content
Entity
Service
Layout