International Journal of Software Engineering
and Knowledge Engineering
Vol. 24, No. 3 (2014) 391–417
© World Scientific Publishing Company
DOI: 10.1142/S0218194014500156

NMMp: A Model-Driven UML Extension for the Description of Navigation Maps for Web Applications

Humberto Cortés* and Antonio Navarro†
DISIA, Universidad Complutense de Madrid
C/Profesor José García Santesmases, s/n
Madrid 280240, Spain
**hjcortes@estumail.ucm.es*
†anavarro@fdi.ucm.es

Received 10 May 2013
Revised 15 August 2013
Accepted 29 January 2014

With the advent of multitier and service-oriented architectures, the presentation tier is more detached from the rest of the web application than ever. Moreover, complex web applications can have thousands of linked web pages built using different technologies. As a result, the description of navigation maps has become more complex in recent years. This paper presents NMMp, a UML extension that: (i) provides an abstract vision of the navigation structure of the presentation tier of web applications, independently of architectural details or programming languages; (ii) can be automatically transformed into UML-WAE class diagrams, which can be easily integrated with the design of the other tiers of the web application; (iii) encourages the use of architectural and multitier design patterns; and (iv) has been developed according to OMG standards, thus facilitating its use with general purpose UML CASE tools in industry.

Keywords: UML profile; meta-object facility; Query/View/Transformation; MDD; Web engineering.

1. Introduction

The design of user interfaces has become more and more complex in the context of web applications, with hundreds of web pages linked by thousands of links. At present, it is not uncommon to have web applications with a mix of static and dynamic pages. Furthermore, with the integration of different platforms these dynamic pages can be built using different technologies: *J2EE* [1], *ASPX pages* [2], *PHP scripts* [3], etc.

The web engineering community has provided *navigation maps* to deal with the complexity of the presentation tier of web applications. Navigation maps present a global view of a web application for an audience [4]. A navigation map describes the possible sequences of web pages displayed to a user, and is typically part of the documentation of a web application [5]. At present, many web sites include navigation maps to help users during browsing, which makes their description a key issue during the development of web applications [6]. Using navigation maps, developers can obtain a global view of the whole application, which can help them during the development process. In addition, the presence of navigation maps can help the users of web sites to find the information they want much more quickly [7].

At present most web engineering notations provide an integral treatment for the design of a web application, binding the navigation map structure to the design of the other tiers. This approach has the advantage of providing an easy way to describe web applications as a whole, facilitating the generation of running web applications [8–11].

However, this approach does not take into account the use of well-defined architectural design patterns for multitier and SOA architecture, which are widely used in industry [12–15]. These patterns affect the design of every tier and provide design rules for the definition and interconnection of these tiers.

It seems, therefore, that there are two approaches when designing web applications: (i) to use a well-known web engineering design notation and, using its specific CASE tool, generate the web application; and (ii) to use a well-known catalogue of web architectural patterns and, using a generic CASE tool, generate part of the web application. In this paper we refer to the first solution as the *notation-based approach* and refer to the second solution as the *pattern-based approach*.

Looking at the sources of both approaches, it is evident that academy is nearer to the notation-based approach [8–11, 16], while industry is nearer to the pattern-based approach [12–15].

However, what is most noticeable in these approaches is their orthogonal development. Thus, most notation-based approaches do not mention architectural multitier and SOA patterns. Conversely, multitier and SOA patterns do not take web engineering notation into account when designing web applications.

We might conceive both approaches at two different levels: notation-based approaches describe the abstract designs of web applications, while architectural patterns describe the detailed designs of web applications. However, when web applications are built at the end, both design and code have to be produced. Consequently: (i) if a notation-based approach is followed, the design is built using a concrete design notation and no presence of any architectural multitier based on the SOA pattern is guaranteed; but (ii) if a pattern-based approach is followed, the most reasonable design notation is UML (most architectural design patterns are described using UML [12, 14, 15], and these patterns are implemented in the running application).

Therefore, at the end of the day, although we can consider notation-based and pattern-based approaches at two different levels, in practice they represent two different philosophies that lead to different developmental processes and applications.

For years, our research group, belonging to academy, has been involved in the development of design notation for hypermedia and web applications [7, 17, 18]. In addition, in recent years, we have been involved with the design, development and deployment of the *Virtual Campus of the Universidad Complutense de Madrid* [19, 20] an e-learning platform devoted to research and academic tasks at our university. As a result of this experience, and notwithstanding our academic origin, we feel nearer to the pattern-based approach than to the notation-based approach.

Pattern-based design provides a flexible and maintainable approach for the design of web applications. The division into tiers of a web application allows us to change one aspect of it (e.g., presentation, logic or integration), minimizing the effects on the other aspects of the application. This maintainability has been tested in industry for years [12–15], while UML diagrams, drawn with generic CASE tools, can be used to represent the design of web applications. This element facilitates the development of design diagrams because UML is a standard notation, often taught in standard courses [21, 22]. In addition, code maintenance is not linked to any specific tool.

In any case, we think that the pattern-based approach can take advantage of specific design notations. In particular, in our opinion, the presentation tier can be a good tier in which to apply a specific design notation. The presentation tiers of web applications have thousands of elements (web pages and hyperlinks) that should be depicted in the most abstract way in order to facilitate their understanding by both customers and developers. Note that customers cannot validate the usefulness of a *business delegate* [12] to hide connection details, but they can and should validate the navigational structure of the web application. In addition, developers need to know the interconnections between web pages in a web application in order to implement them. Therefore, navigation maps are valuable tools for all the stakeholders involved in the development of web applications.

Following this philosophy, we developed NMM (*Navigation Maps Modeling*), a design notation for the description of navigation maps of web applications [7]. This paper presents NMMp as the evolution of NMM notation in terms of a *UML profile* [23] and a set of *QVT operational mapping transformations* [24] that allow NMMp abstract diagrams to be automatically transformed into detailed *UML Web Application Extension* (UML-WAE) diagrams [25]. These UML diagrams can be easily integrated with the design of the other tiers in the web application, using a UML CASE tool, thus enabling a full model-driven architecture approach.

Therefore, NMMp combines the notation-based and the pattern-based approaches. Like the notation-based approach, NMMp provides an abstract vision of the navigational structure of the presentation tier of web applications, independently of architectural details and programming. In line with the pattern-based approach, the UML-WAE class diagrams automatically generated from the NMMp diagrams use the presentation tier's design patterns, while other business and integration patterns can be used in the design of the remaining tiers. In this way, NMMp diagrams are used as architecture-independent *platform-independent models* (PIMs) [26], while the UML WAE diagrams generated are used as architecture-dependent PIMs, which are easier to translate into *platform-specific models* (PSMs) [26]. Thus NMMp is envisioned as a complement to plain UML, which has become the *de facto* standard for describing the business logic, integration and resource tiers [12, 14, 15].

Because NMMp is a specific notation and also uses and promotes the use of architectural patterns, it is referred to as a *mixed-approach* between the notation-based and pattern-based approaches.

Finally, NMMp has been developed using OMG standards, which facilitates its use with commercial and open CASE tools in industry. In particular, we have used the *Borland Together* CASE tool [27] because of its integrated support for XMI [28] and QVT operational mappings. The support for XMI and QVT enables an easy transition from *Borland Together* to other CASE tools supporting these standards.

NMMp's predecessor, NMM, was mainly a formal approach that permitted the definition of *browsing semantics* [29] for web applications. Thus, NMM models can fit reasonably well into the view of *hyperdocuments as automata* [7, 29]. In this view, model checking can be used to verify that browsing specifications are met by the behavior defined by the automaton view of the hyperdocument (*the links-automaton*). However: (i) the visual notation attached to the NMM formal definition was not UML-based and was not related to any other well-known visual notation; (ii) because it was not based on a proprietary standard, NMM needed a proprietary CASE tool; and (iii) the transformations to other models had to be performed manually. Therefore, in practice, and although very well formalized, NMM was not very usable. NMMp has been designed taking into account its usability, instead of its formal properties. Thus, whilst in NMMp the most appealing changes seem to be the changes made in the visual notation, the deepest changes are made in the underlying formal model.

NMM defines a *computing function* [7] for an application A as $comp_A$: $Anchor_c \times Input \rightarrow Page \times AI$. The detailed definition of this function is beyond the scope of this paper, but basically, it takes an anchor able to start a computing process as well as an input value and returns the target page along with six different sets of anchors (hidden behind the set AI [7]) that characterize the static and dynamic anchors always present in this target page or present as a result of the computing itself. This complex information was necessary to provide formal browsing semantics for the application. This formal function imposed important restrictions on the visual diagrams. NMMp, on the other hand, is not so formal and permits the informal definition of the sets of anchors present in a generated web pages (indeed, there are almost no restrictions on this aspect). Therefore NMMp diagrams are easily defined and more intuitive, although they do not have the formal properties that permitted the definition of NMM browsing semantics. We have given NMM's formal properties up for NMMp's UML-based characterization, which permits its use in UML CASE tools and its automatic transition to UML-WAE diagrams using QVT operational mapping transformations.

Thus, NMMp: (i) introduces some modifications to NMM's underlying model in order to make it more usable; (ii) develops NMM from a formal-based notation to a

UML extension, increasing its applicability; (iii) changes the visual appearance of the notation; (iv) implements automatic transformations between NMMp and UML WAE using a commercial CASE tool, thus providing a professional development environment for industry; and (v) defines notation-based, pattern-based and mixed approaches for the development of web applications, making NMMp one of the best representatives of the mixed approach.

In this paper Sec. 2 describes the NMMp profile. Section 3 describes the QVT operational mapping transformations between NMMp and UML WAE. Section 4 compares NMMp with other approaches and, finally, Sec. 5 presents conclusions and future work.

2. NMMp Notation

NMMp is described using a UML profile. This section shows both the metamodel of NMMp and its profile description. NMMp divides the description of navigation maps into three components: (i) page diagrams, which describe the web pages and their navigation links; (ii) region diagrams, which describe the regions which make up a window; and (iii) mixing diagrams, where pages and links in page diagrams are assigned to regions in region diagrams.

The definition of a specific graphical notation for NMMp according to its metamodel was considered, instead of the use of a UML profile that extends UML. However, the UML profile version was chosen because: (i) it allows simpler integration with other UML diagrams; and (ii) it avoids the need for additional tools that support the visual aspect of the graphical notation and its integration with a general purpose CASE tool.

2.1. *Page diagrams*

In NMMp, as in the case of UML WAE, *pages* are considered any component that can be provided by a web server using HTTP requests [25, 30]. In NMMp there are two types of page, which are represented as stereotyped classes: (i) *lasting pages*, which represent static pages (`Lasting Page` stereotype); and (ii) *transient pages*, which represent dynamically generated pages (`Transient Page` stereotype).

NMMp pages can have anchors. Anchors are navigation devices able to generate an HTTP request. In NMMp there are three types of anchor, which are represented as stereotyped classes: (i) *retrieval anchors*, which permit the retrieval of static pages (`Retrieval Anchor` stereotype); (ii) *form computing anchors*, which represent the sending of information using *forms* [31] to computational components that generate transient pages (`Form Computing Anchor` stereotype); and (iii) *non-form computing anchors*, which represent the invocation of computational components that generate transient pages (`Non Form Computing Anchor` stereotype).

NMMp anchors are assigned to NMMp pages using the UML composition relationship, although they are usually depicted as nested classes of these pages.

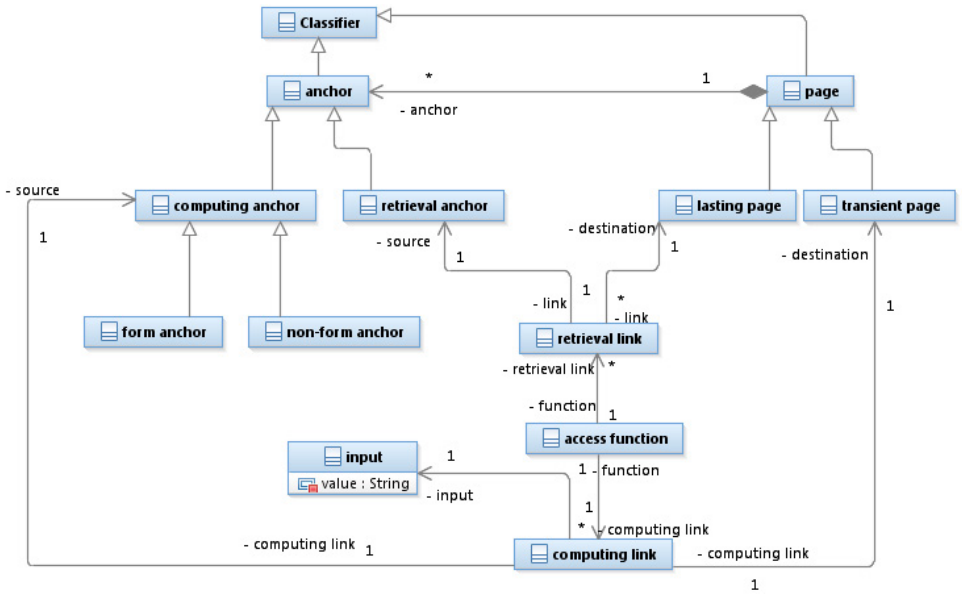


Fig. 1. MOF metamodel for NMMp page diagrams. For the sake of conciseness `Class` and `Association` are defined in UML `Core::Constructs` and merged in the NMMp metamodel.

In NMMp navigation relationships are described as links between anchors and pages. In NMMp there are two types of link, which are represented as stereotyped navigated associations: *lasting page retrieval* (`Retrieval Function` stereotype); and (ii) *transient page generation* by computational devices, e.g. servlets, (`Computing Function` stereotype). Figure 1 depicts the MOF metamodel for NMMp page diagrams.

In order to provide a compact definition of this MOF metamodel, the NMMp metamodel makes a package merge of UML `Core::Constructs` [23].

The metamodel defined in Fig. 1 has been represented as a UML profile in order to extend UML notation. Figure 2 depicts this profile definition using the *Borland Together* CASE tool.

Using this profile, NMMp page diagrams can be defined as a UML extension. Figure 3 depicts an NMMp diagram that describes navigation by the AACV Virtual Campus[a], the latest virtual campus developed by our research group, its design being an evolution of the UCM Virtual Campus.

Figure 3 depicts a `Selector` page that gives a user access to his `Courses`, `Login` (after logout), `ProfileDetails` and `Announcements`. The page `Selector` includes non-form computing anchors (`toCourses`, `closeSession`, `toProfileDetails` and `toAnnouncements`) that give access to these pages. The `Courses` page contains as many anchors (`toCourse`), giving access to detailed course information, as the user

[a] Accessible at http://solaris.fdi.ucm.es/ with user `demo` and key `demoKey`.

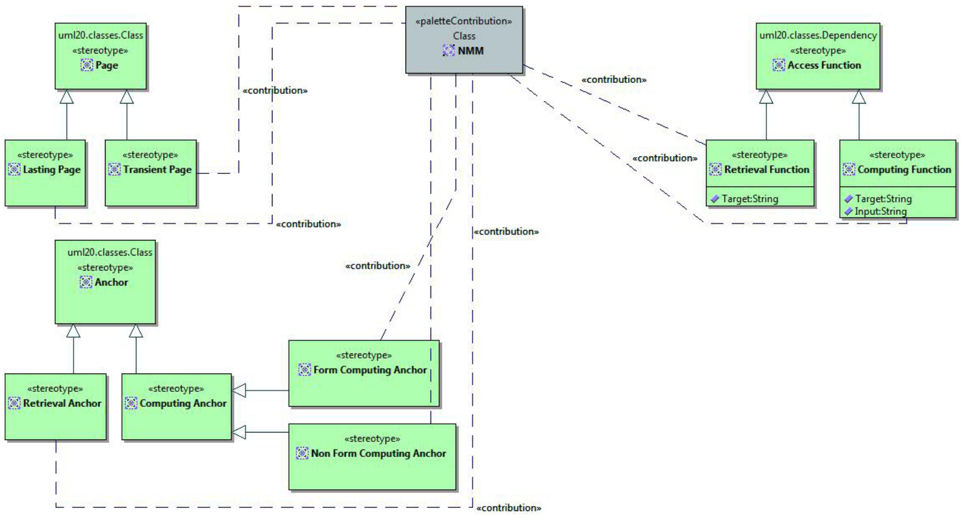
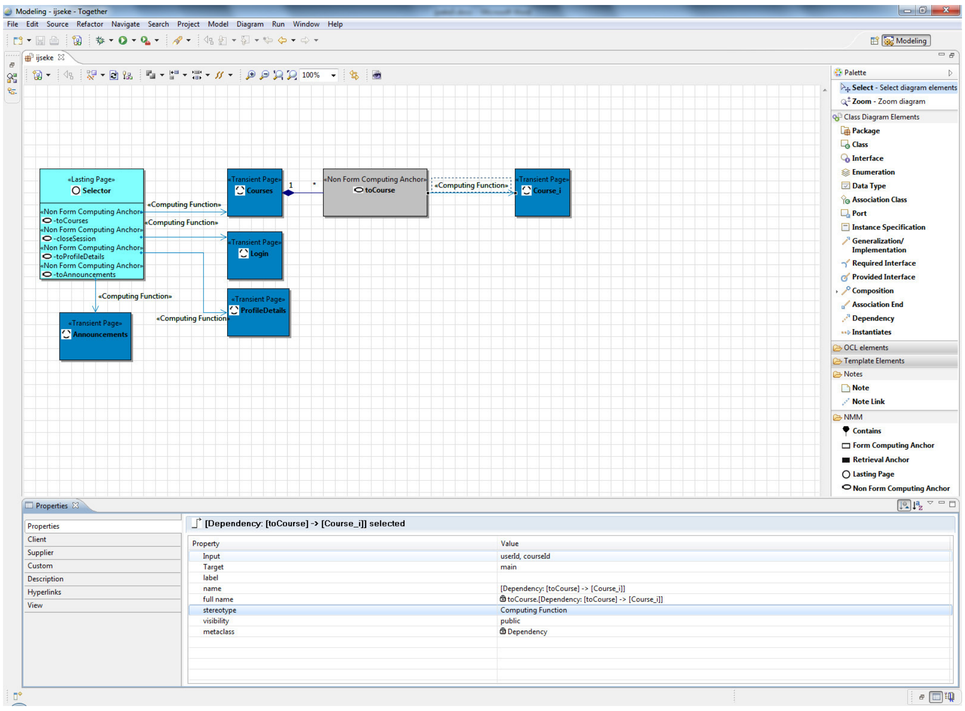


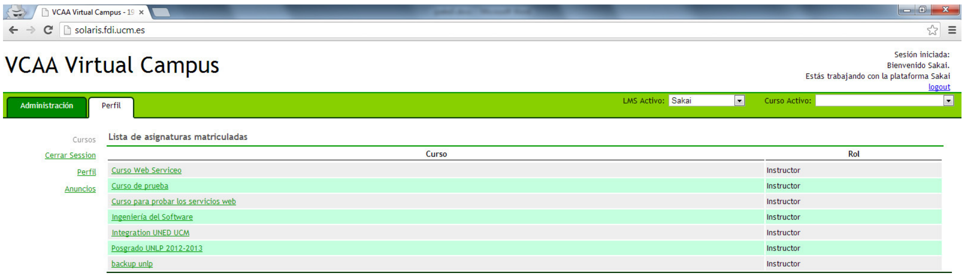
Fig. 2. UML profile for NMMp page diagrams as defined in *Borland Together*.



[Dependency: [toCourse] -> [Course_i]] selected

Property	Value
Input	userId, courseId
Target	main
label	
name	[Dependency: [toCourse] -> [Course_i]]
full name	toCourse.[Dependency: [toCourse] -> [Course_i]]
stereotype	Computing Function
visibility	public
metaclass	Dependency

Fig. 3. NMMp page diagram built with *Borland Together*. The lasting page `Selector` has four nested non-form computing anchors that give access to four transient pages. One of these pages, `Courses`, has an undefined number of `toCourse` anchors which give access to the specific course using the `Input` attribute of the computing links `userId` and `courseId`, as the figure shows.



VCAA Virtual Campus - 19

solaris.fdi.ucm.es

VCAA Virtual Campus

Sesión iniciada:
Bienvenido Sakai.
Estás trabajando con la plataforma Sakai
logout

Administración | Perfil

LMS Activo: Sakai   Curso Activo:

Cursos
Cerrar Session
Perfil
Anuncios

Lista de asignaturas matriculadas

Curso	Rol
Curso Web Serviceo	Instructor
Curso de prueba	Instructor
Curso para probar los servicios web	Instructor
Ingeniería del Software	Instructor
Integration UNED UCM	Instructor
Posgrado UNLP 2012-2013	Instructor
backup unlp	Instructor

Fig. 4. AACV virtual campus with the `Selector` page (left) and the user `Courses` displayed after the selection of the anchor `toCourses`, (named *Cursos* in this figure).

has courses. As Fig. 3 details, the `toCourse` anchor includes `userId` and `courseId` input in the computing function that selects the specified course. Figure 4 depicts a screenshot of this virtual campus with the pages `Selector` and `Courses`.

This is one of the main differences between NMM and NMMp. NMM needs a more complex and formal definition for the anchors defined in a specific page, while NMMp follows the more informal UML-like notation.

As can be seen in Fig. 3, NMMp page diagrams are abstract diagrams, independent of any software architecture and programming language, which depict the pages and their navigation relationships in the presentation tier of a web application.

2.2. *Region diagrams*

In NMMp a window can be divided into different regions using a region diagram. NMMp uses stereotyped classes in order to represent *windows* and *regions*. Aggregation relationships between windows and regions are represented using UML composition relationships.

Figure 5 depicts the MOF metamodel for NMMp region diagrams.

As in the case of page diagrams, in order to provide a compact definition of this MOF metamodel, the NMMp metamodel makes a package merge of UML `Core::Constructs`.

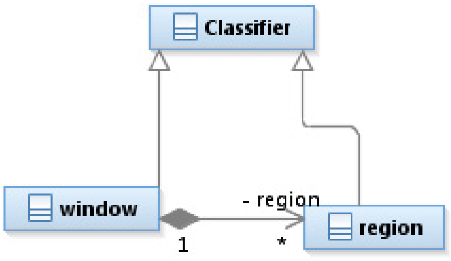


Fig. 5. MOF metamodel for NMMp region and mixing diagrams. For the sake of conciseness `Classifier` is defined in UML `Core::Constructs` and merged in the NMMp metamodel.

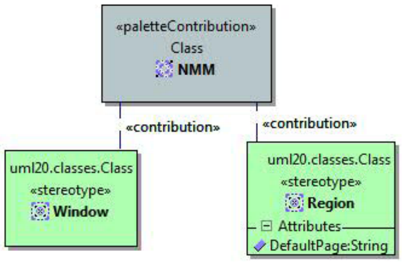
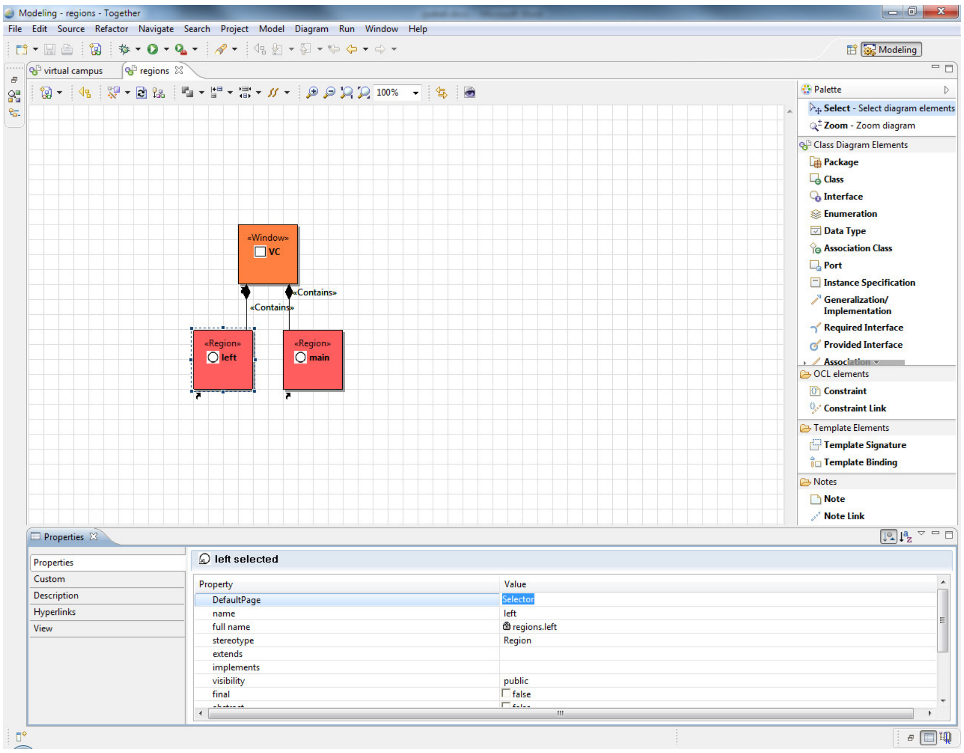


Fig. 6. UML profile for NMMp region diagrams as defined in *Borland Together*.

The metamodel defined in Fig. 5 has been represented as a UML profile in order to extend UML notation. Figure 6 depicts this profile definition using the *Borland Together* CASE tool.

Figure 7 depicts a region diagram that splits the window in two regions: `left` and `main`.



left selected

Property	Value
DefaultPage	Selector
name	left
full name	regions.left
stereotype	Region
extends	
implements	
visibility	public
final	☐ false

Fig. 7. NMMp region diagram describing one window (`vc`) with two regions (`left` and `main`).

2.3. *Mixing process*

In NMMp, page and region diagrams are related in the *mixing process*. This process has two components: (i) regions can have assigned default pages; and (ii) anchors, belonging to pages, can have regions assigned. The first component identifies the page that a region has to load when its window is loaded. The second component works as the `target` attribute of an HTML anchor inside a specific page [31], defining the frame where a link destination has to be displayed. Note that these are simple HTML-based semantics, which in NMMp are not restricted to the `target/frameset` approach.

Figure 8 shows the enrichment of the diagrams in Figs. 1 and 5 with the elements needed for the mixing process. In the NMMp profile this mixing process is achieved using the `DefaultPage` attribute included in every region (Fig. 6), and the `Target` attribute included in every link (Fig. 2).

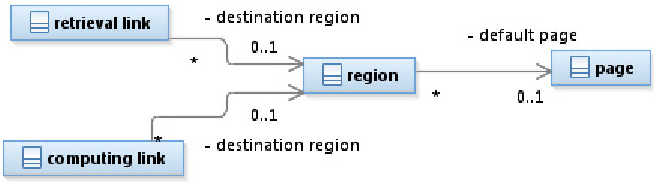


Fig. 8. MOF metamodel for NMMp default page and destination region components. This diagram enriches the diagrams in Figs. 1 and 5.

For example, Fig. 3 depicts how the link that relates the `toCourse` anchor to the page `Course_i` has assigned main as a destination region using its attribute `Target`. This would depict the page `Course_i`, the destination of the `toCourse` anchor in the `main` region. In Fig. 7 the `left` region has the `Selector` page assigned as a value of its `DefaultPage` attribute.

3. QVT Transformations

In NMMp QVT operational mapping transformations have been developed in order to allow NMMp abstract diagrams to be transformed automatically into detailed UML WAE diagrams.

UML WAE is very useful for depicting the presentation tier of multitier applications [7, 25]. In UML WAE contents provided by the web server are called *pages*. Pages are represented by stereotyped classes. There are different types of page: (i) static web pages, represented by the stereotype `client page`; (ii) computational processes running in the server (e.g. *servlets* [1]), represented by the stereotype `server page`; and (iii) forms, represented by the stereotype `form`. Navigation relationships between pages are represented by stereotyped navigated associations. There are different types of relationship: (i) hyperlinks, represented by the stereotype `link`; (ii) data sending between forms and computational processes, represented by the stereotype `submit`; (iii) forwarding of control between computational processes, represented by the stereotype `forward`; and (iv) construction of static web pages by computational processes, represented by the stereotype `build`.

3.1. *Transformation rationale*

Following the notation-based approach we have defined NMMp notation for describing navigation maps for web applications. NMMp diagrams are high-level design diagrams, very useful to both customers and developers for defining the navigation structure of a web application. However, NMMp diagrams are very far from the pattern-based approach, where specific architectural patterns can be defined. Thus, their translation to UML WAE models, which are closer to architectural patterns, is very valuable.

NMMp QVT operational mapping transformations are based on NMM transformation rules for automatically translating NMMp diagrams into UML WAE diagrams, following Model 1 or Model 2 architecture [32]. They are summarized in Table 1. Because NMM diagrams were not UML-based and the transformations had to be performed manually, the applicability of NMM was seriously handicapped. However, in NMMp these rules are implemented as QVT operational mapping

Table 1. Transformation rationale between NMMp and UML WAE for Model 1 and Model 2 architectures.

NMMp element	UML WAE element. Model 1	UML WAE element. Model 2
lasting page	client page	client page
transient page	client page generated by server page	client page generated by server page
retrieval anchor	—	—
non-form computing anchor	—	—
form computing anchor	form page	form page
link with source in retrieval anchor	link between pages	link between pages using controller
link with source in non-form computing anchor	link to a server page + transfers + hook for business logic invocation (i.e. façade or business delegate)	link to output server pages using controller + transfers + hook for business logic invocation (i.e. façade or business delegate).
link with source in form-computing anchor	submit to server page + transfers + hook for business logic invocation (i.e. façade or business delegate)	submit to output server pages using controller + transfers + hook for business logic invocation (i.e. façade or business delegate).
window	frameset	
region	target	
aggregation from window to region	aggregation from frameset to target	
default page assignation	value assignment to DefaultPage attribute of target	
destination region	value assignment to Target attribute of corresponding link	

transformations, enabling automatic transformation and, thus enhancing its applicability to commercial UML CASE tools.

Basically:

- NMMp pages are translated into UML WAE pages.
- NMMp anchors are ignored, because they have no counterpart in UML WAE, except for form computing anchors, which are translated into form pages. However, NMMp computing anchors are necessary for the definition of the computational artifacts involved in their processing.
- Links are translated into navigated association between pages. In Model 1 these associations are made directly between pages, while in Model 2, these associations are made via a *controller* [12]. In addition, computing anchors are related to business tier devices able to describe business logic invocation. These devices can be *façades* and *application services* [12] for those applications without distributed business logic, or *business delegates* [12] for those applications with distributed business logic, including remote objects or web services. The controller invokes these components using *commands* [33], as in the case of the *Struts* [34] framework. Data is moved between tiers using *transfers* [12].
- Windows and regions are transformed into framesets and frames.
- Default page assignations are transformed into aggregation relationships between frames and pages.
- The destination region is transformed into the value assigned to the target attribute of a link.

The following subsections describe the QVT operational mapping transformation for the rules depicted in Table 1. For the sake of conciseness, only transformations to Model 2 are included.

3.2. *QVT operational mapping transformations*

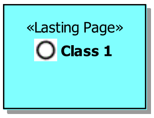
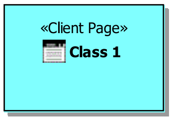
3.2.1. *Page diagrams translation*

This section describes the QVT Operational Mapping transformations for translating page diagrams into UML WAE diagrams.

Figure 9 describes the transformation rule for translating NMMp lasting pages into UML WAE pages. Basically, NMMp lasting pages are translated into UML WAE client pages.

Figure 10 describes the transformation rule for translating NMMp transient pages into UML WAE pages. Basically, NMMp transient lasting pages are translated into UML WAE client pages generated by server pages. In this case, three transformations are needed.

Figure 11 describes the transformation rule for translating NMMp retrieval anchors into UML WAE pages. Basically, NMMp retrieval anchors are translated into UML WAE connections via the controller. Again, three transformations are needed.

NMMp lasting page	**UML WAE client page**
	

QVT operational mapping transformation

```
mapping uml20::classes::Class::ToClientPage() :
        uml20::classes::Class
        when
        {
                self.stereotypes = OrderedSet{'Lasting Page'}
        }
        {
                object
                {
                        name := self.name;
                        description := self.description;
                        stereotypes := OrderedSet { 'Client Page' };
                }
        }
```

Fig. 9. Translation for NMMp lasting pages.

Figure 12 is a visual representation of the transformation rule for translating NMMp computing anchors into UML WAE pages, as depicted in Fig. 13. In this case, the UML WAE pages are connected via the controller, and the controller invokes the business façade using commands/actions. In addition, other elements of the business tier (transfers and applications services) are generated. In this case four transformations are needed. The first transformation only applies to form computing anchors.

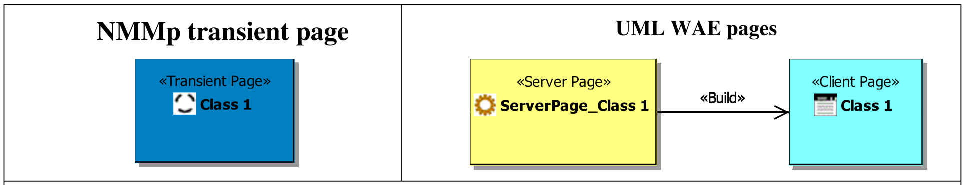
3.2.2. *Translation of region diagrams*

This section describes the QVT Operational Mapping transformations for translating region diagrams into UML WAE diagrams.

Figure 14 describes the transformation rule for translating windows and regions into UML WAE pages. Basically, NMMp windows and regions are transformed into UML WAE framesets and targets. Of course, transformation rules can be provided for other HTML presentation elements, such as regions built according to the `div` tag [31].

3.2.3. *Mixing process*

Finally, transformations are needed in order to implement the mixing process. Using these transformations, default pages are assigned to UML WAE target elements, and links are assigned to these targets. Figure 15 depicts these transformations.



QVT operational mapping transformations

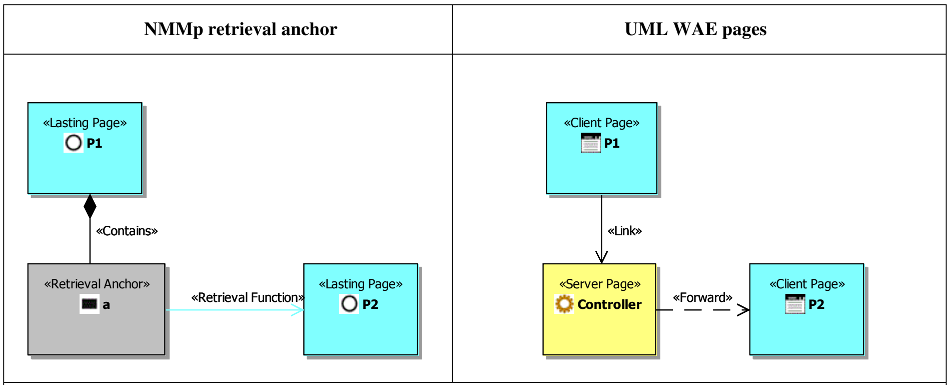
```
mapping uml20::classes::Class::ToClientPage() : uml20::classes::Class
        when
        {
                self.stereotypes = OrderedSet{'Transient Page'}
        }
        {
                object
                {
                        name := self.name;
                        description := self.description;
                        stereotypes := OrderedSet { 'Client Page' };
                }
        }

mapping uml20::classes::Class::ToServerPage() : uml20::classes::Class
        when
        {
                self.stereotypes = OrderedSet{'Transient Page'}
        }
        {
                object
                {
                        name := 'ServerPage_' + self.name;
                        description := self.description;
                        stereotypes := OrderedSet { 'Server Page' };
                }
        }

mapping uml20::classes::Class::ToBuild() : uml20::classes::Dependency
        {
                object
                {
                        client      := self.resolve(uml20::classes::Class)->
                                                 select(c | c.stereotypes->
                                               includes('Server Page'));
                        supplier    := self.resolve(uml20::classes::Class)->
                                                 select(c | c.stereotypes->
                                  includes('Client Page'));
                        stereotypes := OrderedSet { 'Build' };
                }
        }
```

Fig. 10. Translation for NMMp transient pages.

3.3. *Example*

Sections 3.1 and 3.2 have introduced and defined QVT transformations for translating NMMp diagrams into UML diagrams. In this section a simple example is given.

If we apply the transformations defined in Sec. 3.2 to the NMMp diagram depicted in Fig. 3, the UML WAE diagram depicted in Fig. 16 is obtained.



QVT operational mapping transformations

```
query uml20::classes::Class::HasBeenConvertedToController() : Boolean
    {
        if((self.resolve(uml20::classes::Class)->select(class |
            class.name = 'Controller'))->any() = true) then
            true
        else
            false
        endif
    }
mapping uml::together::Model::ToControllerClass() : uml20::classes::Class
    {
        object
        {
                name := 'Controller';
                stereotypes := OrderedSet{ 'Server Page' };
        }
    }
mapping uml20::classes::Dependency::ToForwardDependencies(in NMMModel : uml::together::Model) :
uml20::classes::Dependency
    {
        init
        {
      var Supplier := self.supplier.oclAsType(uml20::classes::Class);
        result := NMMModel.CreateDependency(Supplier,'Controller', 'Client Page', 'Forward');
        }
}
```

Fig. 11. Translation for NMMp retrieval anchors.

In this transformation, NMMp pages have been translated into UML WAE pages, while NMMp computing anchors have been used to create the computational components of the business tier responsible for their processing. Thus, NMMp computing anchors hide the actions invoked by the controller, the façade (or business delegate) responsible for their implementation (using application services) and the data transfer objects that move data between tiers. For example, the `Courses` page in Fig. 3 has been translated into the pages `toCourses_View`, and `Courses`, while the `toCourse` anchor has been translated into the classes `toCourse_InputTransfer`, `toCourse_OutputTransfer`, `toCourseAction`, `toCourse_AS`, and the function `toCourse` of the façade. Links between anchors and pages in the NMMp diagram have been translated into links between UML WAE elements using the `Controller`.

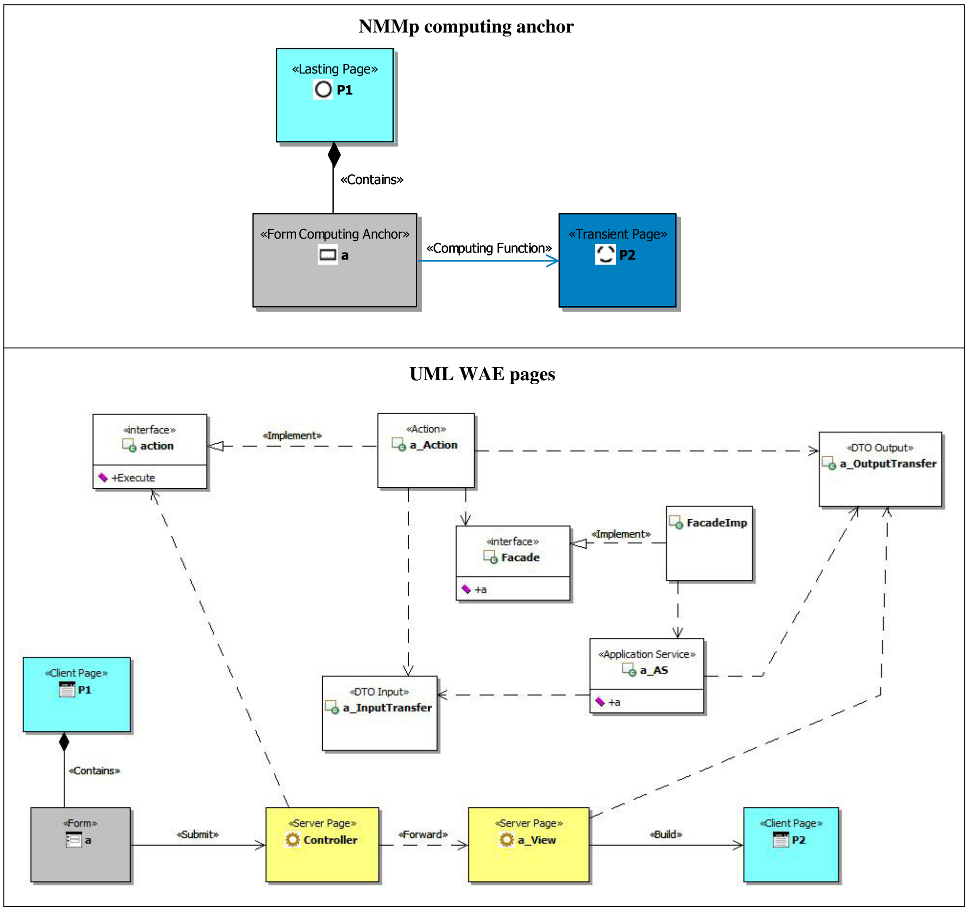


Fig. 12. Translation for NMMp computing anchors (i). Form computing anchors are graphically depicted, but the transformation rules affect all the types of computing anchors. The translation for non-form computing anchors is similar but excludes details such as the Form.

As Fig. 16 shows, UML WAE is only applied at the presentation tier. Because a Model 2 architecture has been chosen, the navigation flow depicted in the NMMp diagram in Fig. 3 is made opaque in Fig. 16. In general, the diagram in Fig. 3 is valuable for both customers and developers, while the diagram in Fig. 16 is valuable only for developers, who can develop the navigation and computational artifacts needed to build the web application. It is well worth noting that the NMMp diagram in Fig. 3 has eleven UML classes (six pages and five anchors) while the UML WAE diagram in Fig. 16 has thirty five classes, demonstrating the semantic power of NMMp elements.

The diagram in Fig. 16 describes the presentation tier in UML WAE. Because this diagram and the NMMp diagram in Fig. 3 are UML diagrams, they can be

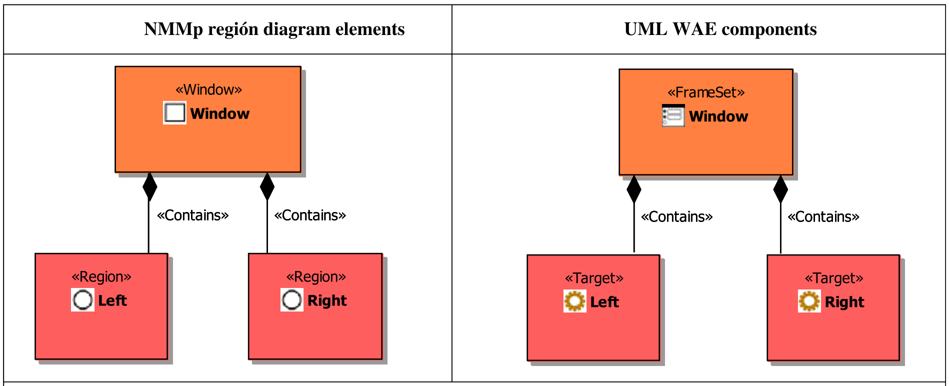
QVT operational mapping transformations

```
mapping uml::together::Model::ToActionClass() : uml20::classes::Class
{
        object
        {
           name := 'action';
           stereotypes := OrderedSet{ 'interface' };
           ownedOperations += object uml20::kernel::features::Operation
                                        { name := 'Execute'; }
        }
}

query uml20::classes::Class::CreateM2BussinessLayerClasses() : Set(uml20::classes::Class)
{
        Set
        {
            self.ToWAEClass(self.name + '_View', 'Server Page', ''),

            self.ToWAEClass(self.name + '_Action', 'Action', ''),

            self.ToDTOClass(self.name + '_InputTransfer', 'DTO Input'),

            self.ToDTOClass(self.name + '_OutputTransfer', 'DTO Output'),

            self.ToApplicationService()
        }
}

query uml20::classes::Class::CreateM2Dependencies(in NMMModel : uml::together::Model) :
Set(uml20::classes::Dependency)
{
        Set
        {
           NMMModel.CreateDependency(self, 'FacadeImp',
                                     'Application Service'),

           NMMModel.CreateDependency(self, 'Controller',
                                     'Server Page', 'Forward'),

           self.CreateDependency(NMMModel, 'Action', 'Facade'),

           self.CreateDependency(NMMModel, 'Action', 'action',
                                           'Implement'),

           self.ToDTOOutputDependencies()
        }
}
```

Fig. 13. Translation for NMMp computing anchors (ii).

integrated with other UML diagrams drawn with a general purpose UML CASE tool (*Borland Together* in our case). Several multitier patterns are used in this translation (application controller, transfers, application services) and other multitier and SOA patterns can be used to describe the remaining tiers in the application.

In this way, the NMMp approach encourages the use of architectural patterns for the design of web applications, as industry does, at the same time using navigation maps to increase the abstraction level of the design, as favored by academy. Thus, a balance between pattern-based and notation-based approaches is achieved in NMMp.



QVT operational mapping transformation

```
mapping uml20::classes::Class::ToFrameset() : uml20::classes::Class
        when
        {
        self.stereotypes = OrderedSet{Window'}
        }
        {
                object
                {
                        name := self.name;
                        description := self.description;
                        stereotypes := OrderedSet { 'FrameSet' };
                }
        }

mapping uml20::classes::Class::ToTarget() : uml20::classes::Class
        when
        {
        self.stereotypes = OrderedSet{Region}
        }
        {
                object
                {
                        name := self.name;
                        description := self.description;
                        stereotypes := OrderedSet { 'Target' };
                }
        }
```

Fig. 14. Translation for NMMp region diagrams.

Finally, Fig. 17 depicts the UML WAE translation for the NMMp region diagram in Fig. 7.

Regarding the mixing process and the default page assignation, note how in Fig. 17 the `DefaultPage` attribute of the target `left` has the `Selector` page as value. Although for the sake of conciseness the destination region assignment is not depicted in Fig. 16, the value of the `Target` attribute of the link that connects the `Courses` page and the `Controller` has the value `main`, as was defined in Fig. 3.

```
mapping uml20::classes::Class::ToTarget(inout Target
uml20::classes::Class):uml20::classes::Class
        when { self.stereotypes = OrderedSet{'Region'} }
        {       init
                {       Target.name := self.name;
                        Target.description := self.description;
                        Target.stereotypes := OrderedSet { 'Target' };
                        Target.setPropertyValue('DefaultPage',
        self.getPropertyValue('DefaultPage'));
                        result := Target;
                }
        }
mapping uml20::classes::Class::ToLink(inout Link : uml20::classes::Dependency) :
uml20::classes::Dependency
        { init
                { Link.client := self.dependencies->select(d | d.stereotypes->
        includes('Contains'))-> collect(dc |
        dc.client.oclAsType(uml20::classes::Class))
                  Link.supplier := self.dependencies->select(d |d.stereotypes-
        >includes('Retrieval
                                                    Function'))->collect(dc |
                           c.supplier.oclAsType(uml20::classes::Class));
                  Link.stereotypes := OrderedSet{ 'Link' };

        Link.setPropertyValue('Target',self.getPropertyValue('Target'));
                }
}
```

Fig. 15. Translation for NMMp mixing process.

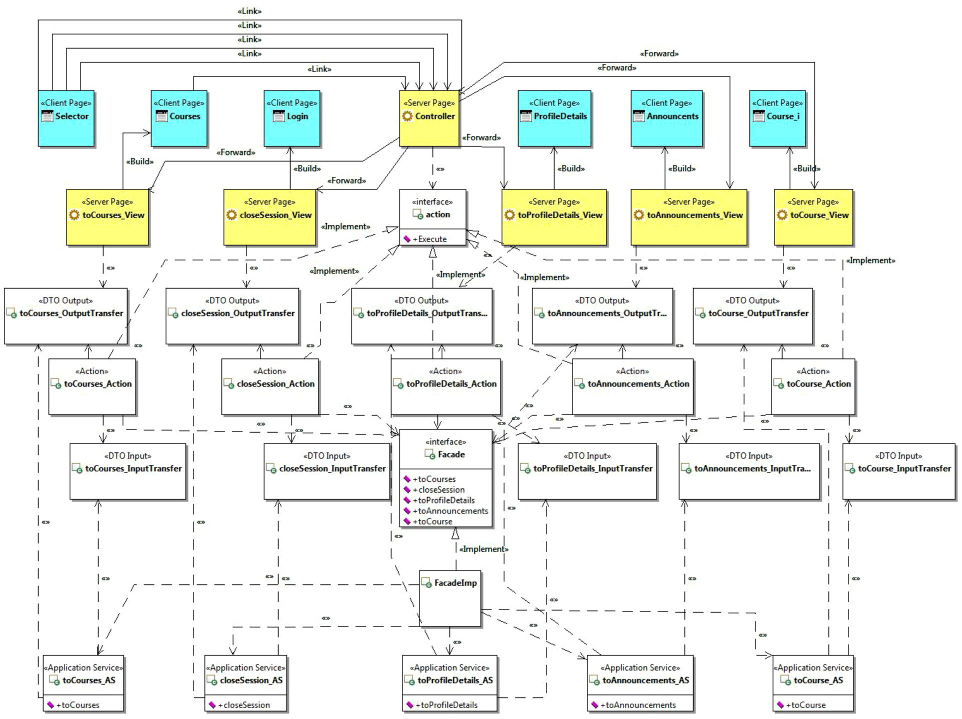
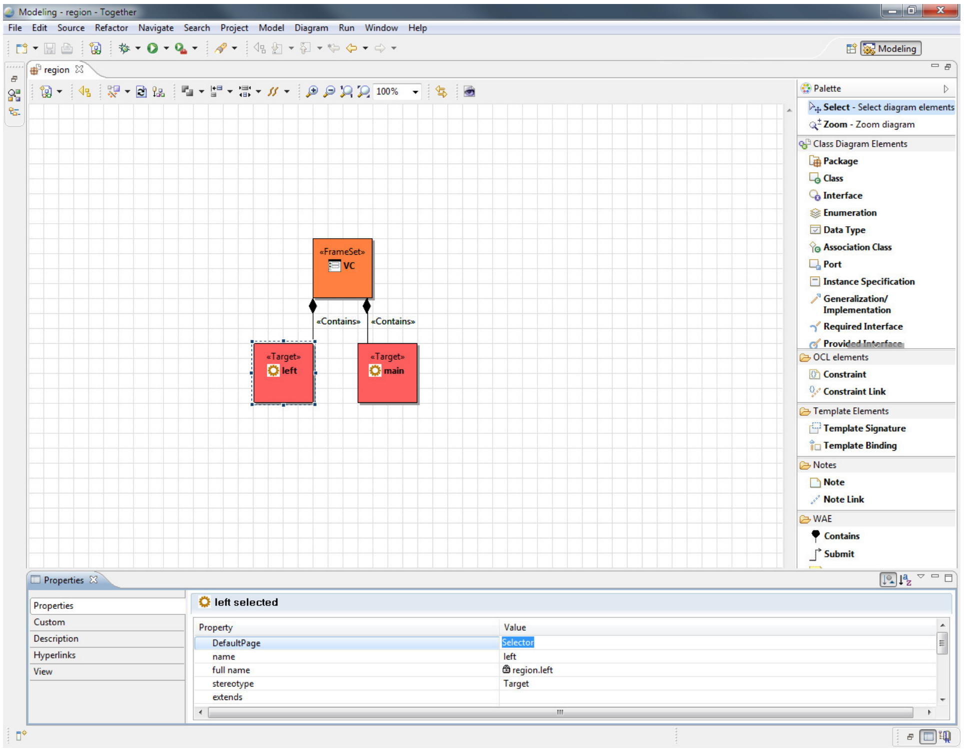

«Link»
«Link»
«Link»
«Link»
«Link»
«Forward»
«Forward»
«Client Page» Selector
«Client Page» Courses
«Client Page» Login
«Server Page» Controller
«Client Page» ProfileDetails
«Client Page» Announcents
«Client Page» Course_i
«Forward»
«Build»
«Forward»
«Build»
«Forward»
«Build»
«Build»
«Build»
«Server Page» toCourses_View
«Server Page» closeSession_View
«interface» action
+Execute
«Server Page» toProfileDetails_View
«Server Page» toAnnouncements_View
«Server Page» toCourse_View
«Implement»
«Implement»
«Implement»
«Implement»
«Implement»
«DTO Output» toCourses_OutputTransfer
«DTO Output» closeSession_OutputTransfer
«DTO Output» toProfileDetails_OutputTrans...
«DTO Output» toAnnouncements_OutputTr...
«DTO Output» toCourse_OutputTransfer
«Action» toCourses_Action
«Action» closeSession_Action
«Action» toProfileDetails_Action
«Action» toAnnouncements_Action
«Action» toCourse_Action
«interface» Facade
+toCourses
+closeSession
+toProfileDetails
+toAnnouncements
+toCourse
«DTO Input» toCourses_InputTransfer
«DTO Input» closeSession_InputTransfer
«DTO Input» toProfileDetails_InputTransfer
«DTO Input» toAnnouncements_InputTra...
«DTO Input» toCourse_InputTransfer
«Implement»
FacadeImp
«Application Service» toCourses_AS
+toCourses
«Application Service» closeSession_AS
+closeSession
«Application Service» toProfileDetails_AS
+toProfileDetails
«Application Service» toAnnouncements_AS
+toAnnouncements
«Application Service» toCourse_AS
+toCourse

Fig. 16. UML WAE Model 2 diagram automatically generated from the NMMp page diagram in Fig. 3.



left selected

Property	Value
DefaultPage	Selector
name	left
full name	region.left
stereotype	Target
extends	

Fig. 17. UML WAE diagram automatically generated from the NMMp region diagram in Fig. 7.

4. Related Work

This section analyzes NMMp, comparing it with different design notations. These notations are grouped in four levels of abstraction:

— *Level IV: PSM-based.* In this category, design diagrams are platform-specific models, and they are not derived from platform-independent models (PIMs). This category includes notations used by tools such as: *Oracle ADF* [35] and the web diagrams generated with *IBM Rational Software Architect* [36]. Note that *IBM Rational Software Architect* is a general purpose UML CASE tool. However, it includes specific design notations for the description of web applications.

— *Level III: architecture-dependent PIM-based.* In this category PIMs can be defined, but these PIMs include specific details of Model 1 or Model 2 architecture. This category includes notations such as: plain *UML* [37], and *UML WAE* [25].

— *Level II: PIM-based.* In this category architecture-independent PIMs are provided, but they are translated into architecture-dependent PIM and PSM models. This category includes notations such as: *NMMp*, *OOH4RIA* [11], and *UWA* [9].

— *Level I: only PIM-based.* In this category independent PIMs are provided, but they are directly translated into code. Therefore, no other PIMs or PSMs are provided. This category includes notations such as: *RUX* [16], *UWE* [10] and *WebML* [8].

In Table 2, these notations are analyzed, taking into account ten characteristics:

— *Presence of architecture-independent PIMs.* This characteristic is very valuable because it allows an abstract view of the application to be provided, facilitating user validation [7, 8, 10, 11, 16].
— *Presence of additional PIMs and/or PSMs.* This characteristic is important because it provides additional semantics for the abstract PIMs, enhancing the maintainability of the code by programmers [7, 9, 11].
— *Description of other tiers in addition to presentation tier.* A multitier web application is comprised of several tiers. The more tiers described by an approach, the more integrated views of the application are provided by the approach, and the more valuable the approach is considered [8–11].
— *Compatibility with explicit use of architectural multitier* and *SOA patterns.* The use of these patterns enhances the comprehensibility and maintainability of the applications [12–15]. Therefore, their presence is customary in enterprise applications that have to be maintained over years.

Table 2. Comparison of NMMp with other approaches.

Abstraction level	Approach	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)
I. Only PIM-based	RUX	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗
	UWE	✓	✗	✓	✗	✓	✓	✗	✓	✓/×	✗
	WebML	✓	✗	✓	✗	✓	✓	✓/×	✓	✓/×	✗
II. PIM-based	NMMp	✓	✓	✓/×	✓	✗	✓	✓	✗	✓	✓
	OOH4RIA	✓	✓	✓	✓/×	✓	✓	✗	✓	✗	✗
	UWA	✓	✓	✓	✓/×	✗	✓	✗	✓	✗	✗
III. Arch.-dependent PIM-based	UML WAE	✗	✓	✓/×	✓	✗	✓	✗	✗	✓	✓
	UML	✗	✓	✓	✓	✓	✓/×	✗	✗	✓	✓
IV. PSM-based	IBM RSA	✗	✗	✓	✓/×	✓/×	✓	✗	✓	✗	✓/×
	Oracle ADF	✗	✗	✓	✓/×	✓	✓	✗	✓	✗	✗

(i) Presence of architecture-independent PIMs.
(ii) Presence of additional PIMS and/or PSMs.
(iii) Description of other tiers in addition to presentation tier.
(iv) Compatibility with explicit use of architectural multitier and SOA patterns.
(v) Description of RIA user interface widgets.
(vi) Description of navigation flow.
(vii) Hiding of computational artifacts.
(viii) Generation of running application.
(ix) Supported by general purpose UML CASE tool.
(x) Code maintenance independent of specific development environments.

— *Description of Rich Internet Application (RIA) user interface widgets.* RIA user interface widgets permit enhanced interactions for users. Therefore, they are increasingly popular among developers and designers [8, 10, 11, 16].
— *Description of navigation flow.* Navigation flow helps customers, designers and programmers to use, design and build more usable web applications, making it an important component of design notations [7, 8, 10, 11, 16].
— *Hiding of computational artifacts.* The explicit presence of computational artifacts (e.g. *commands* of an *application controller* [12]) hinders the comprehensibility of the navigation flow by customers and designers [7]. Therefore, their exclusion enhances the comprehensibility of web notations.
— *Generation of running application.* Some approaches include tools that permit the generation of the running application, mixing design diagrams and code components. This gives added value to design diagram validation [8, 10, 11, 35, 36], but links code maintenance to a specific tool.
— *Support by a general purpose UML CASE tool.* Every notation can build its own CASE tool. However, the use of a general purpose CASE tool facilitates the use of the notation, unlinking maintenance of the notation from a particular tool [7, 10].
— *Code maintenance independent of specific development environments.* The binding of code maintenance to a specific development environment is a problem, as it requires the code to be linked to an individual tool [38]. Maintenance of code built without design diagrams is a very difficult task [38, 39]. However, the presence of abstract design diagrams which are more application-oriented than code-oriented can make both the design and maintenance processes difficult [40]. Therefore, dependence on a specific tool which provides a direct translation from abstract PIMs to code should be avoided, if we wish to make code maintenance independent of particular tools.

Level I approaches are abstract notations that, with the exception of RUX, cover all the aspects of web applications. However the abstraction of their design diagrams hinders the direct implementation of these diagrams. Therefore, the approaches include specific development environments that permit code generation, binding its maintenance to the generation tool of the development environment. UWE and WebML can be used with general purpose UML CASE tools but this greatly limits their capabilities for application generation. In addition, the use of multitier and SOA architectural patterns is not present in these approaches. Thus, for example, in WebML, data movement between layers is accomplished using *XML elements* [41] instead of transfer objects. Level I approaches are the prime example of *notation-based approaches* for the development of web applications.

Level II approaches try to translate abstract PIMs into more specific PIMs and PSMs. In OOH4RIA the translation to PSMs affects only diagrams of the presentation tier. Therefore, navigation diagrams still remain very abstract, being directly transformed into code. In this way, the maintenance drawbacks of the Level I approach are still present. In UWA every abstract PIM is translated into a PSM but, as in the case of OOH4RIA, the use of architectural multitier and SOA architectural patterns is not described or encouraged and is, therefore, not guaranteed in these approaches (although OOH4RIA mentions some of them in an example). In addition, both approaches are linked to specific development environments. NMMp PIMs only describe elements of the presentation tier that are translated into architecture-specific UML WAE PIM diagrams that include several architectural patterns. These diagrams can be integrated with plain UML diagrams that describe the design of the other tiers. In this way, any pattern of business, integration and resource tiers can be described in UML and integrated with UML WAE diagrams generated from NMMp notation. In addition, NMMp can be used in a general purpose UML CASE tool with support for QVT transformations, such as *Borland Together*. Thus, NMMp tries to find a tradeoff between notation-based and pattern-based development, which can be defined as a mixed approach. The main drawback is that NMMp does not include the description of RIA widgets. They have not been initially included because their presence forces the presence of more specific web clients, provoking possible compatibility problems between browsers, as in the case of other client-based technologies [12, 14]. However, the inclusion of RIA widgets in NMMp is currently planned. Finally, in NMMp, code generation is restricted to that provided by the general purpose UML CASE tool, or the transformations provided by the user, which do not usually generate running applications.

Level III approaches provide architecture-dependent PIMs. Thus the UML WAE diagrams lack abstraction [8], becoming very complex for Model 2 applications [7]. However, UML WAE diagrams can be used in combination with NMMp diagrams to increase their level of abstraction, at the same time depicting the architectural details needed by programmers. It is well worth mentioning that, in a multitier design, UML WAE diagrams are only used for the presentation tier [7]. Plain UML can also be used to describe the presentation layer but in this case navigation relationships have to be provided, using component or artifact diagrams [37, 42], which hinders the integrated view of the design of the web application, represented in terms of physical elements (i.e. components or artifacts) for the presentation tiers, and logical elements (i.e. classes) for the remaining tiers. As UML diagrams include architectural multitier and SOA patterns, the use of plain UML is, nowadays, the best exponent of the pattern-based development of web applications [12, 14, 15]. As in the case of NMMp, code generation is restricted to that provided by the general purpose UML CASE tool, or the transformations provided by the user, which usually do not generate running applications.

Finally, Level IV approaches are dependent on specific technologies. Thus IBM RSA provides design diagrams to build J2EE and Struts applications. Oracle ADF provides a complete visual environment for the building of J2EE applications, using almost every technology for this platform. However, IBM RSA and Oracle ADF diagrams are platform-specific, and very tied to J2EE technology. (Note that, as was previously mentioned, *IBM Rational Software Architect* is a general purpose UML CASE tool. However, it includes specific design notations for the description of web applications.) In this sense they are nearer to visual programming environments than to CASE tools. In addition, the inclusion of multitier and SOA patterns is restricted to those supported by the tools.

5. Conclusions and Future Work

The design of web applications is a complex issue, as evidenced by the presence of various notations and other tools for their design and development.

As a general rule there are powerful approaches that cover every aspect of web applications, using specific abstract design notations. These notations are not code-oriented, but code is obtained using a specific tool. In this way, application maintenance is linked to a specific notation and tool. Normally, no architectural design patterns are depicted or encouraged in these approaches, which have been called notation-based in this paper.

On the other hand, there are commercial tools, nearer to visual programming environments for specific languages, which use visual diagrams to configure the code for web applications, although in this case the approach is very far from abstract design notations. In addition, in most cases, application maintenance is linked to a specific tool, and the application of architectural patterns is subordinated to their inclusion in the tool.

There are two intermediate approaches. One is based on the use of plain UML and architectural multitier and SOA patterns for the development of web applications. This approach has been called pattern-based in this paper. However, plain UML is not enough to provide a detailed description of the presentation tier of web applications.

The other intermediate approaches use abstract notations, and convert them into PSMs to facilitate code generation. As the PSMs are defined and code is generated using open transformations, code maintenance is not linked to any tool. Thus, if the approach allows the inclusion of architectural multitier and SOA patterns, software maintenance is enhanced. NMMp falls into this category, which this paper calls the mixed approach.

Although each approach has its advantages and drawbacks, the NMMp approach can claim to be a balanced notation for the representation of navigation maps for web applications.

According to our experience of designing and building web applications, UML is enough to describe the design of the business, integration and resource tiers of web applications. Therefore, only a complement for the presentation tier of web applications is needed. UML WAE could be used, but its design diagrams, although very valuable for programmers because they include architectural details, can become very complex for representing the navigation structure of web applications.

NMMp notation is used to complement UML WAE diagrams for representing navigation maps, abstracting their nature, and making them suitable for validation by customers. NMMp diagrams are abstract, independent of architectural details or programming languages, and are thus valuable for both customers and developers. In addition, NMMp diagrams and their UML WAE translations are intended to be used in the presentation tier, complementing plain UML diagrams for the other tiers. Generated UML WAE diagrams include architectural patterns, and plain UML diagrams can include the remaining multitier and SOA patterns. This allows an integrated use of NMMp and UML notation, using and promoting architectural patterns in every tier. Furthermore, because NMMp is a UML-based notation, the diagrams of all tiers of web applications can be produced using a general purpose UML CASE tool with support for XMI and QVT transformations. Finally, the code is obtained using transformation rules, explicitly defined by the user, or included in a general purpose CASE tool. Thus, although the generated code is not as complete as that generated by notation-based tools, it is more maintainable, and not tied to a specific notation or tool.

However, NMMp does not have RIA capabilities and it does not provide the running application, as most notation-based approaches do. RIA capabilities could be included, as in other design notations, but the client-nature of RIA interfaces has curbed their inclusion in NMMp up to now. In any case, NMMp notation focuses on navigation maps, and a combination of RUX and NMMp could be used for a detailed description of the presentation widgets of RIA applications, as in [43].

Regarding running applications, NMMp considers design diagrams as abstract representations of code, not as visual representations of code [42, 44]. Therefore, in the NMMp approach, code can be generated, but not the whole application.

Future work includes the completion of QVT transformations with *OCL constraints* [45] to ensure the syntactic validity of NMMp diagrams, and the inclusion of RIA widgets in NMMp design primitives. In this connection, the integration of RUX and NMMp should be analyzed. We are also working on the inclusion of security roles and permissions attached to pages, in order to include the definition of security constraints in NMMp. Finally, a mixed approach is currently being applied to generating additional components in other tiers of web applications. In this work, the configuration of the designs with a wide range of architectural patterns is being considered.

Acknowledgments

Ministerio de Ciencia e Innovación (project AACV TIN2009-14317-C03-01), and Universidad Complutense de Madrid (group 921340) have supported this work.

References

1. J. L. Weaver, K. Mukhar and J. P. Crume, *Beginning J2EE 1.4: From Novice to Professional* (Apress. 2004).
2. I. Spaanjaars, *Beginning ASP.NET 4 in C# and VB* (Wrox, 2010).
3. E. Lecky-Thompson and S. D. Nowicki, *Professional PHP* (Wrox, 2009).
4. S. M. Abraháo, L. Olsina and O. Pastor, Towards the quality evaluation of functional aspects of operative web applications, in *Proc. IWCMQ*, 2002.

5. M. Han and C. Hofmeister, Separation of navigation routing code in J2EE web applications, in *Proc. ICWE 2005*, Sydney, Australia, 2005.
6. A. Navarro, J. L. Sierra, A. Fernández-Valmayor and B. Fernández-Manjón, Conceptualization of navigational maps for web applications, in *Proc. MDWE 2005*, Sydney, Australia, 2005.
7. A. Navarro, A. Fernández-Valmayor, B. Fernández-Manjón and J. L. Sierra, Characterizing navigation maps for web applications with the NMM approach, *Science of Computer Programming* **71**(1) (2008) 1–16.
8. A. Bozzon, S. Comai, P. Fraternali and G. Toffetti, Conceptual modeling and code generation for rich internet applications, in *Proc. ICWE 2006*, 2006.
9. D. Distante, P. Pedone, G. Rossi and G. Canfora, Model-driven development of web applications with UWA, MVC and JavaServer faces, in *Proc. ICWE 2007*, 2007.
10. N. Koch, A. Kraus, C. Cachero and S. Meliá, Integration of business processes in web application models, *Journal of Web Engineering* **3**(1) (2004) 22–49.
11. S. Meliá, J. Gómez, S. Pérez, and O. Díaz, Architectural and technological variability in rich internet applications, *IEEE Internet Computing* **14**(3) (2010) 24–32.
12. D. Alur, J. Crupi and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies. 2nd ed.* (Prentice Hall/Sun Microsystems Press, 2003).
13. T. Erl, *SOA Design Patterns* (Prentice Hall PTR, 2009).
14. M. Fowler, *Patterns of Enterprise Application Architecture* (Addison-Wesley Professional, 2002).
15. P. B. Monday, *Web Service Patterns: Java Edition* (Apress, 2003).
16. M. Linaje, J. C. Preciado and F. Sánchez-Figueroa, Engineering rich internet application user interfaces over legacy web models, *IEEE Internet Computing* **11**(6) (2007) 53–59.
17. A. Navarro, B. Fernández-Manjón, A. Fernández-Valmayor and J. L. Sierra, Formal-driven conceptualization and prototyping of hypermedia applications, in *Proc. FASE 2002*, 2002.
18. A. Navarro, A. Fernández-Valmayor, B. Fernández-Manjón and J. L. Sierra, Conceptualization, prototyping and process of hypermedia applications, *International Journal of Software Engineering and Knowledge Engineering* **14**(6) (2004) 565–602.
19. A. Navarro and A. Fernández-Valmayor, Conceptualization of hybrid web sites, *Internet Research* **17**(2) (2007) 207–228.
20. A. Navarro, J. Cristobal, C. Fernández-Chamizo and A. Fernández-Valmayor, Architecture of a multiplatform virtual campus, *Software: Practice and Experience* **42**(10) (2012) 1229–1246.
21. ACM/IEEE Computer Society, *Software Engineering 2004* (http://sites.computer.org/ccse/, 2004).
22. IEEE Computer Society, *Guide to the Software Engineering Body of Knowledge 2004 Version* (http://www.computer.org/portal/web/swebok, 2004).
23. OMG, *UML Infrastructure Specification Version 2.3* (http://www.omg.org/spec/UML/2.3/, 2010).
24. OMG, *Meta Object Facility (MOF) 2.0 Query/View/Transformation, V1.0* (http://www.omg.org/spec/QVT/1.0/, 2008).
25. J. Conallen, *Building Web Applications with UML, 2nd ed.* (Addison-Wesley Professional, 2002).
26. OMG, *MDA Guide Version 1.0.1* (http://www.omg.org/cgi-bin/doc?omg/03–06–01, 2003).
27. Borland, *Together*, (http://www.borland.com/us/products/together/, 2013).
28. OMG, *XML Metadata Interchange (XMI)* (http://www.omg.org/spec/XMI/, 2007).

29. P. D. Stotts, R. Furuta and C. Ruiz, Hyperdocuments as automata: Verification of trace-based browsing properties by model checking, *ACM Transactions on Information Systems* **16** (1998) 1–30.
30. L. Shklar and R. Rosen, *Web Application Architecture: Principles, Protocols and Practices* (Wiley, 2009).
31. W3C, *HTML 4.01 Specification* (http://www.w3.org/TR/html4/, 1999).
32. S. Brown *et al.*, *Professional JSP, 2nd ed.* (Wrox, 2001).
33. E. Gamma, R. Helm, R. Johnson and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley Professional, 1994).
34. D. Brown, C. M. Davis and S. Stanlinck, *Struts 2 in Action* (Manning Publications, 2008).
35. Oracle, *Application Development Framework (ADF)* (http://www.oracle.com/technetwork/developer-tools/adf/overview/index.html, 2013).
36. IBM, *Rational Software Architect* (http://www.ibm.com/developerworks/rational/products/rsa/, 2013).
37. J. Arlow and I. Neustadt, *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design, 2nd ed.* (Addison-Wesley Professional, 2005).
38. R. S. Pressman, *Software Engineering: A Practitioner's Approach, 7th ed.* (McGraw-Hill, 2009).
39. I. Sommerville, *Software Engineering, 9th ed.* (Addison-Wesley. 2010).
40. C. Barry and M. Lang, A survey of multimedia and web development techniques and methodology usage, *IEEE Multimedia* **8**(3) (2001) 52–60.
41. W3C, *Extensible Markup Language (XML), 5th ed.* (http://www.w3.org/TR/2008/REC-xml-20081126/, 2008).
42. G. Booch, R. A. Maksimchuck, M. W. Engel and B. J. Young, *Object-Oriented Analysis and Design with Applications, 3rd ed.* (Addison-Wesley Professional, 2007).
43. J. C. Preciado, M. Linaje, R. Morales-Chaparro, F. Sánchez-Figueroa, G. Zhang, C. Kroiss and N. Koch, Designing rich internet applications combining UWE and RUX-method, in *Proc. ICWE 2008*, 2008.
44. J. Rumbaugh, I. Jacobson and G. Booch, *Unified Modeling Language Reference Manual* (Addison-Wesley Professional, 2004).
45. OMG, Object *Constraint Language, Version 2.2* (http://www.omg.org/spec/OCL/, 2010).