

UWE Metamodel and Profile

User Guide and Reference

Technical Report 0802
Programming and Software Engineering Unit (PST)
Institute for Informatics
Ludwig-Maximilians-Universität München, Germany
February 2008



The UWE Metamodel and Profile – User Guide and Reference
Version 1.0 - February 2008

Christian Kroiß and Nora Koch

Ludwig-Maximilians-Universität München (LMU), Germany
Institute for Informatics
Programming and Software Engineering (PST)
www.pst.ifi.lmu.de/projekte/uwe

This research has been partially supported by the project MAEWA “Model Driven Development of Web Applications” (WI841/7-1) of the Deutsche Forschungsgemeinschaft (DFG), Germany and the EC 6th Framework project SENSORIA “Software Engineering for Service-Oriented Overlay Computers” (IST 016004).

Table of Contents

1	INTRODUCTION	5
2	REQUIREMENTS PACKAGE	6
3	CONTENT PACKAGE	6
4	NAVIGATION PACKAGE.....	6
4.1	CLASS DESCRIPTIONS	7
4.1.1	<i>Node</i>	7
4.1.2	<i>Link</i>	7
4.1.3	<i>NavigationClass</i>	8
4.1.4	<i>NavigationProperty</i>	8
4.1.5	<i>NavigationLink</i>	9
4.1.6	<i>Menu</i>	9
4.1.7	<i>AccessPrimitive</i>	10
4.1.8	<i>Index</i>	10
4.1.9	<i>Query</i>	11
4.1.10	<i>GuidedTour</i>	11
5	PRESENTATION PACKAGE.....	12
5.1	CLASS DESCRIPTIONS	13
5.1.1	<i>PresentationElement</i>	13
5.1.2	<i>PresentationClass</i>	13
5.1.3	<i>PresentationProperty</i>	14
5.1.4	<i>Page</i>	14
5.1.5	<i>PresentationGroup</i>	15
5.1.6	<i>UIElement</i>	15
5.1.7	<i>UIContainer</i>	16
5.1.8	<i>Form</i>	16
5.1.9	<i>AnchoredCollection</i>	16
5.1.10	<i>Anchor</i>	16
5.1.11	<i>Button</i>	17
5.1.12	<i>Text</i>	17
5.1.13	<i>Image</i>	17
5.1.14	<i>TextInput</i>	18
5.1.15	<i>Choice</i>	18
6	PROCESS PACKAGE.....	18
6.1	CLASS DESCRIPTIONS	20
6.1.1	<i>ProcessClass</i>	20
6.1.2	<i>ProcessLink</i>	21
6.1.3	<i>ProcessProperty</i>	21
6.1.4	<i>UserAction</i>	21
7	UWE PROFILE.....	22
8	EXAMPLE: SIMPLE MUSIC PORTAL.....	24
8.1	USE CASES	24
8.2	CONTENT MODEL	25
8.3	USER MODEL.....	26
8.4	NAVIGATION MODEL.....	27
8.5	BUSINESS PROCESSES	27
8.5.1	<i>Process Login</i>	29
8.5.2	<i>Process Logout</i>	30
8.5.3	<i>Process BuyAlbum</i>	30
8.5.4	<i>Process Register</i>	31
8.5.5	<i>Process Recharge</i>	32
8.6	PRESENTATION MODEL	33

1 Introduction

Web modelling approaches are driven by the separation of concerns describing a web system, such as content, hypertext structure, presentation, and processes. The UML-based Web Engineering (UWE) approach provides a set of web domain-specific model elements for modelling these different concerns. These model elements and the relationships between them are specified by a metamodel.

The UWE metamodel is defined as a conservative extension of the UML 2.0 metamodel. Conservative means that the model elements of the UML metamodel are not modified. Instead, all new model elements of the UWE metamodel are related by inheritance to at least one model element of the UML metamodel. We define additional features and relationships for these new elements. Analogous to the well-formedness rules in the UML specification, we use OCL constraints to specify the additional static semantics of these new elements.

The resulting UWE metamodel is profileable, which means that it is possible to map the metamodel to a UML profile [1]. In particular, UWE stays MOF-compatible, i.e. UWE is compatible with the MOF interchange metamodel and therefore with tools that are based on the corresponding XML interchange format XMI. The advantage is that all standard UML CASE tools, which support UML profiles or UML extension mechanisms can be used to create UWE models of, web applications. If technically possible, these CASE tools can further be extended to support the UWE method, i.e. steps of automatic generation of models. ArgoUWE and MagicDraw present instances of such CASE tool support for UWE based on the UWE metamodel.

The UWE extension of the UML metamodel consists of adding two top-level packages Core and Adaptivity to the UML (see Figure 1). The separation of concerns of web applications is reflected by the package structure of Core, the crosscutting of adaptation by the dependency of Adaptivity on Core.

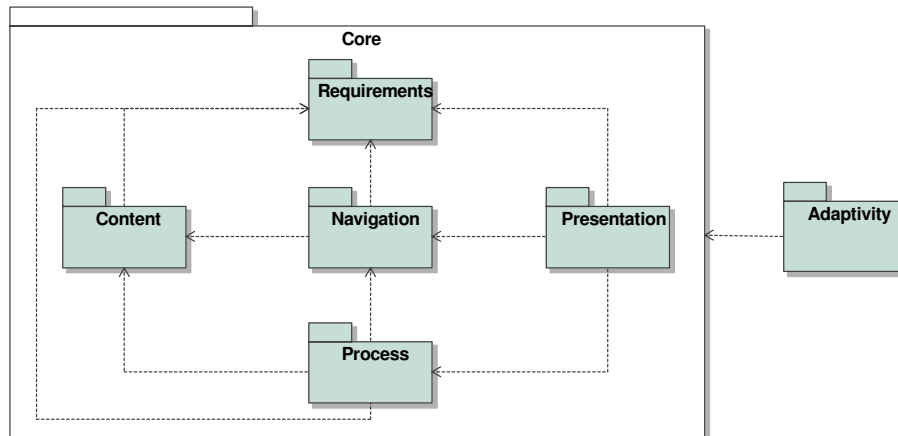


Figure 1: Overview of the UWE Metamodel

2 Requirements Package

The package Requirements comprises the UWE extensions on use case models for discerning navigational from business process and personalized use cases and the extensions for activity diagrams. Further details on these elements will be provided in the next version of this report.

3 Content Package

Content modelling for web applications within UWE does not differ from content modelling for non-web software. Therefore, we use standard UML model elements for structure modelling as classes, associations and packages. In addition, behavioural modelling can make use of UML features such as state machines and sequence diagrams.

Customized web applications require the modelling of user or environment features. A user profile or user model can be used to represent these features separating user profiling from content modelling.

4 Navigation Package

The UWE navigation metamodel is represented in

Figure 2. The backbone of the navigation metamodel is the pair of abstract metaclasses Node and Link and the associations between these classes. A set of subclasses of Node and Link provide the web domain specific metaclasses for building the navigation model: NavigationClass and ProcessClass with the related Navigation Link and ProcessLink as well as Menu and the access primitives Index, GuidedTour and Query.

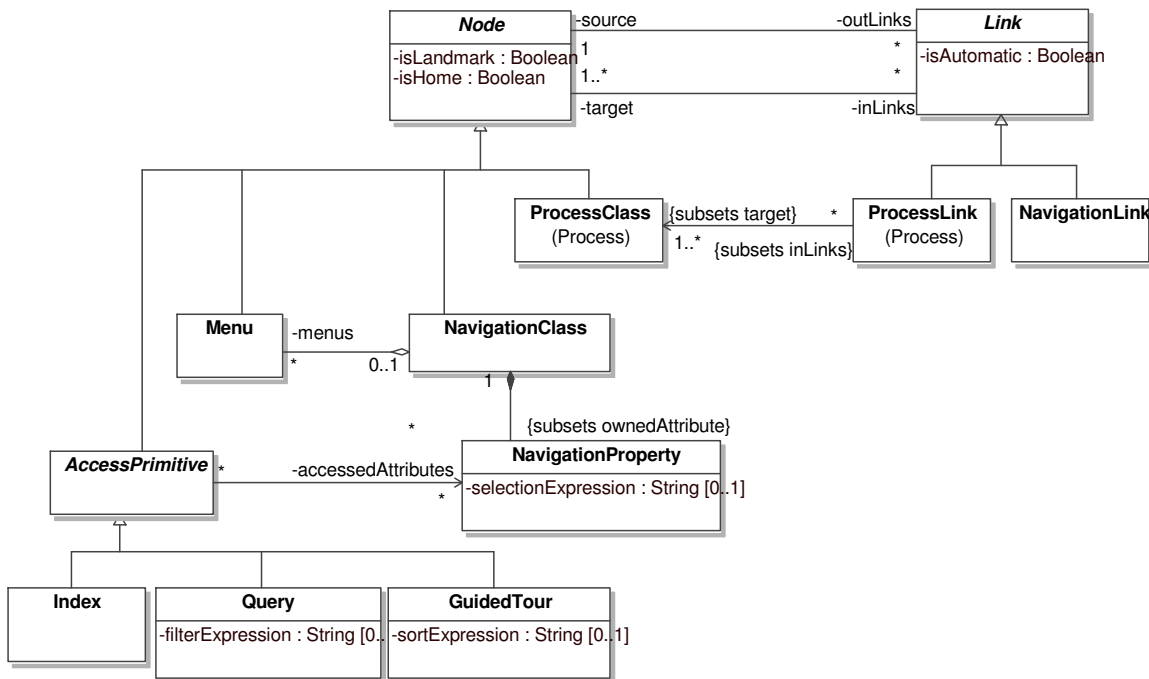


Figure 2: The Navigation Package

The relationship between model elements of the Navigation package and UML classes used to model the content of a web application is shown in Figure 3.

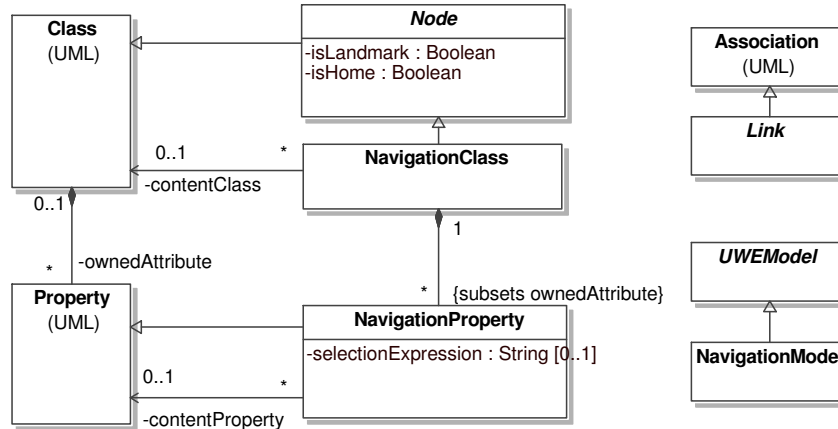


Figure 3: Relationship of the Navigation Package to the UML

4.1 Class Descriptions

4.1.1 Node

Abstractly spoken, a *node* can be any kind of node in a navigation graph. This generally means that when the node is reached during navigation, the user is provided with some information and is optionally offered the possibility to carry out one or more actions.

A *node* does not necessarily represent a page of the web application, although it may do so. What is shown on a page is defined in the presentation model (see section 5).

Generalizations

- *Class* (from UML)

Attributes

- *isLandmark* : Boolean Specifies whether the node is a landmark, which means that it is reachable from every other node of the navigation graph.
- *isHome* : Boolean If this attribute is set to true, the node becomes the origin of the navigation graph.

Associations

- *inLinks* : Link [*] The collection of links that lead to the node.
- *outLinks* : Link [*] The collection of links that originate from the node.

4.1.2 Link

A *link* is an edge of the navigation graph and therefore connects two *nodes*. Note remember that just as a *node* does not always represent a page, a *link* does not have to represent a page transition that is triggered by a user action. As mentioned in section 5, the presentation model defines whether the information of two *nodes* that are connected by a *link* is shown at the

same time or if the user has to click on an anchor to navigate from one node to the other (see section 5 for more information).

Generalizations

- *Association* (from UML)

Attributes

- `isAutomatic` : Boolean This attribute allows specifying explicitly that no decision by the user is required for following the link.

Associations

- `source` : Node [1] The origin node of the link.
- `target` : Node [1..*] The target node(s) of the link. Multiple target nodes are used for adaptivity, which is not described in this version of the document.

4.1.3 NavigationClass

A *navigation class* represents a navigable node of the hypertext structure and establish the connection between the navigation model and the content model. A *navigation class* that is associated with a class from the content model is meant to represent the content of one instance of that class.

Generalizations

- *Node* on page 7.

Attributes

No additional attributes.

Associations

- `contentClass` : Class [0..1] The class of the content model that specifies the content of the *navigation class*.
- `menus` : Menu [*] The collection of all menus that are directly reachable from the *navigation class*, i.e. menus that are targets of *navigation links* that originate from the *navigation class*.
- `navigationProperty` :
NavigationProperty [*]
{subsets ownedAttribute} The collection of *navigation properties* that define the contents of the *navigation class*.

4.1.4 NavigationProperty

The attributes of a *navigation class* are called *navigation properties*. They define the content of *UI elements*. The value of a navigation property is either directly taken from an associated property of a content class or derived using a selection expression. At the moment, UWE does not specify any special syntax or semantics or which languages can be used for the selection expression.

It is common practice in navigation diagrams to leave out navigation properties that are connected to properties of the content class. Although, they can also be specified explicitly to point out what information is relevant. If a navigation class has no navigation properties at all, then each property of the content class is implicitly mirrored by a “virtual” navigation property with the same name.

Generalizations

- *Property* (from UML)

Attributes

- `selectionExpression : String [0..1]` An expression that has the same type as the *navigation property* and that is used to derive a value from the currently available set of content class instances. The context of the expression (self in OCL) is the content class that is associated with the navigation class.

Associations

- `contentProperty : Property [0..1]` A property whose value is mirrored in the *navigation property*. Typically, this is an attribute of the content class that is associated with the navigation class (like `Album::name` in Figure 8). In any case, the instance of the class that contains the content property must be identifiable clearly when the *navigation class* is reached.

4.1.5 NavigationLink

A *navigation link* is a *link* that connects any kind of *nodes* except *process classes*. If either source or target of a link is a *process class* then a *process link* is used (see section 6.1.2).

Generalizations

- *Link* on page 7.

Attributes

No additional attributes.

Associations

No additional associations.

4.1.6 Menu

A *menu* is used to handle alternative navigation paths. Note that a *menu* in the navigation model is not always rendered as a menu in the sense of user interfaces. This is because two *nodes* that are connected through a menu could be defined to be rendered simultaneously in the presentation model (see section 5). In this case, the user does not have to do anything in order to follow the navigation path, which would be the essential behaviour for a menu in any user interface.

Generalizations

- *Link* on page 7.

Attributes

No additional attributes.

Associations

- navigationClass : NavigationClass [0..1] The *navigation class* that is the origin of all navigation paths through the menu.

4.1.7 AccessPrimitive

Access primitives are used to select the instances of content classes that make up the content of *navigation classes*.

Generalizations

- *Node* on page 7.

Attributes

No additional attributes.

Associations

- accessedAttributes : NavigationProperty [*] A collection of *navigation properties* that are used to select the content class instance.

4.1.8 Index

An *index* allows selecting one content class instance from a set of instances that has been compiled during previous navigation. This means that a set of content class instances is taken from the context of the predecessor in the navigation path and the user is allowed to choose one of them. The chosen instance then becomes the content object for the navigation class that succeeds the index in the navigation path.

The input set of content class instances is determined by the incoming *navigation link*. There are three different cases:

1. The predecessor is a *query*. In this case, the set of instances is just the result of the query, e.g. all books published in December 2007.
2. The predecessor is a *navigation class*. The set of instances is then taken from a collection property of the corresponding content class. To specify which property is used, the target role name of the navigation link can be set equal to the property's name.
3. The predecessor is a *menu*. In this case, the context of the menu's preceding navigation class is used just as if it is connected directly and the rules described in 2 apply. In the Music Portal example at the end of the document, this constellation is shown in Figure 10 on page 27. The link from `UserMenu` to `UserAlbumIndex` has one navigable role named `ownedAlbums`. This means that the input collection of albums for `UserAlbumIndex` is taken from the role `ownedAlbums` of the `User` class from the user model (see section 8.3).

Generalizations

- *AccessPrimitive* on page 10.

Attributes

No additional attributes.

Associations

No additional associations.

4.1.9 Query

A *query* is used to retrieve content from a data source. Unlike an *index*, a *query* does not get its set of content class instances from a predecessor of the navigation path, but rather from a database or any other kind of data source that supports queries. A query may require search parameters, like e.g. for searching movies by title. In this case, the presentation model must contain elements that provide a user interface for filling in values for the parameters (see section 5). The semantics of a query can be specified as follows the attribute `filterExpression` of *Query*, which can hold an expression that describes the query and which involves the `accessedAttributes` (see Figure 10).

If the query does not require parameters, it is executed automatically when it is reached in the navigation graph. An example would be a query that retrieves the current top 10 movies from the movie database.

Generalizations

- *AccessPrimitive* on page 10.

Attributes

- `filterExpression` : String [0..1] An expression that describes the semantics of the query.

Associations

No additional associations.

4.1.10 GuidedTour

A *guided tour* provides sequential process to instances of a *navigation class*. It is given an ordered set of content class instances as input and has an outgoing *navigation link* to a *navigation class*. The user is allowed to browse back and forth through the input collection selecting one instance at a time as content for the target *navigation class*. The order in which the instances are visited is specified using a filter expression.

Generalizations

- *AccessPrimitive* on page 10.

Attributes

- `filterExpression` : String [0..1] An expression that is used to calculate the order in which the navigation class instances are visited.

Associations

No additional associations.

5 Presentation Package

The presentation model provides an abstract view on the user interface (UI) of a web application. It is based on the navigation model. The presentation model abstracts from concrete aspects of the UI, like the use of colours, fonts, and where the UI elements are placed on the web page; instead, the presentation model describes the basic structure of the user interface, i.e., which UI elements (e.g. text, images, anchors, forms) are used to present the navigation nodes (see Figure 4 and Figure 5). Also, the UI elements do not represent concrete components of any presentation technology but rather describe what functionality is required at that particular point in the user interface. This could simply mean that a text or image has to be displayed or for example that the user should be enabled to trigger a transition in the navigation model. In the last case, it is clear that an *Anchor* would be used in the UWE presentation model, but UWE does not define how the anchor should be rendered in the final web application. This could of course be just an anchor element of HTML (<a>), but also a button or even an embedded flash applet could serve the purpose.

The basic elements of a presentation model are the *presentation classes*, which are directly based on nodes from the navigation model, i.e. navigation classes, menus, access primitives, and process classes. Presentation classes can contain other presentation elements. This is accomplished through *presentation properties* that use the included *presentation elements* as type. In the case of UI elements, like text or image, the presentation property is associated with a navigation property that contains the content to be rendered.

The inclusion of *presentation classes* into other *presentation classes* or *pages* leads to a tree of presentation classes that are shown together. This means that the links between their corresponding navigation nodes are effectively “followed automatically”. On the other hand, if two presentation classes do not belong to the same inclusion tree, then the link between their navigation nodes has to be triggered by user action.

In contrast to *presentation classes* and *pages*, a *presentation group* defines a set of *presentation classes* that are shown alternatively, depending on navigation. In the sense of the description above, a presentation group creates a set of alternative inclusion trees.

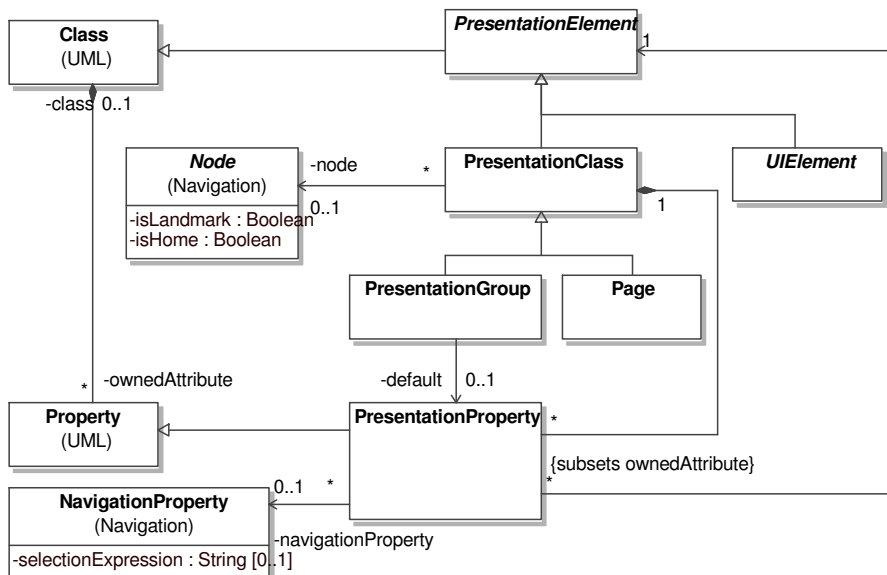


Figure 4: The Backbone of the Presentation Package

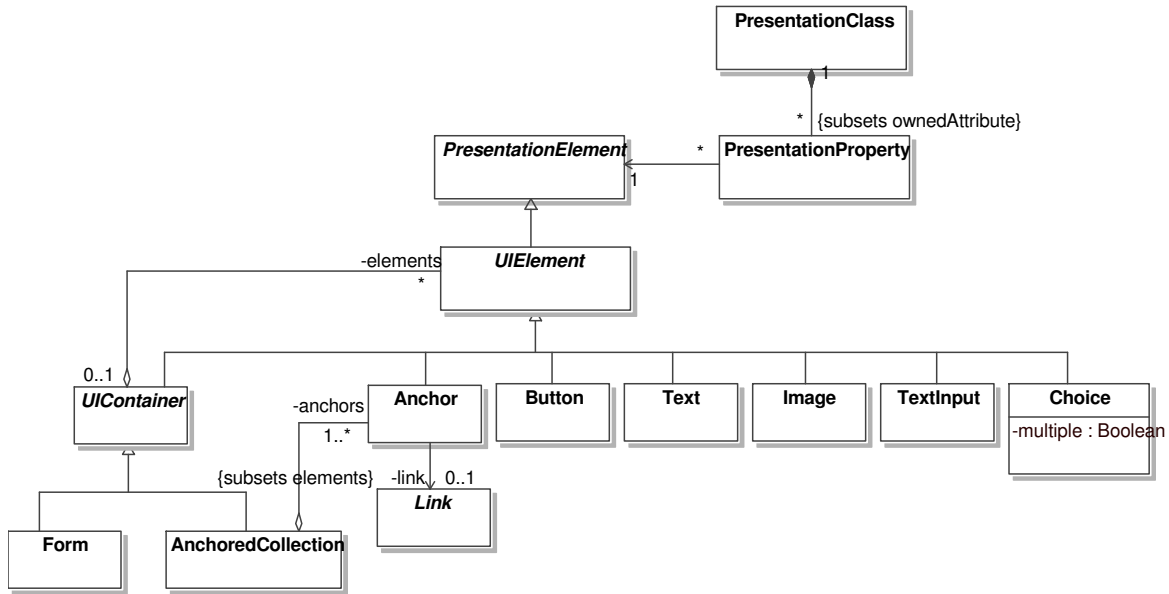


Figure 5: Presentation Elements

5.1 Class Descriptions

5.1.1 PresentationElement

PresentationElement is the abstract super class of all model elements of the presentation package.

Generalizations

- *Class* (from UML)

Attributes

No additional attributes.

Associations

No additional associations.

5.1.2 PresentationClass

A *presentation class* defines the combination of presentation elements that show the contents of a navigation node. If the associated navigation node is reached, the complete composed content of the corresponding inclusion tree is shown.

Generalizations

- *PresentationElement* on page 13.

Attributes

No additional attributes.

Associations

- `node : Node [0..1]` The navigation node that is rendered by the

- | | |
|--|--|
| <ul style="list-style-type: none"> presentationProperty :
PresentationProperty [*]
{subsets ownedAttribute} | <p>presentation class.</p> <p>The collection of presentation properties that constitute the content of the presentation class.</p> |
|--|--|

5.1.3 PresentationProperty

Presentation properties are used to define the content of presentation classes. The presentation element that should be included is used as the type of the *presentation property*. If the contained element is an UI element (like text, image, text input, etc.) then the presentation property can be associated with a navigation or process property that defines the location of data to be presented or edited.

If the property has a multiplicity higher than one, it means that the contained element is rendered repeatedly by iteration over a source collection of values. This collection is given implicitly when the presentation property represents the anchors of an index. Otherwise, the associated navigation or process property must have a multiplicity higher than one, too.

Generalizations

- *Property* (from UML)

Attributes

No additional attributes.

Associations

- | | |
|--|---|
| <ul style="list-style-type: none"> presentationElement :
PresentationElement [1]
{subsets type} | <p>The presentation element that should be included inside the presentation class that owns the presentation property.</p> |
| <ul style="list-style-type: none"> navigationProperty :
NavigationProperty [0..1] | <p>The navigation or process property that defines the location of data that is presented or edited by the included presentation element.</p> |

5.1.4 Page

A *page* has the same semantics as a presentation class, with the exception that it may not be included inside another presentation class. This means that a page always defines the root of an inclusion tree of presentation classes. Unlike a presentation class, a page does not have to be associated with a navigation node, as long as it includes at least one presentation class that provides the reference to the navigation model.

Generalizations

- *PresentationClass* on page 13.

Attributes

No additional attributes.

Associations

No additional associations.

5.1.5 PresentationGroup

A *presentation group* is used to define a set of presentation classes whose contents are shown alternatively on the same area of the page, depending on navigation. If a navigation node is reached that is associated with one of the alternatives, the content of this presentation class replaces the content of the presentation class that is shown at that moment. One of the presentation classes can be defined as the default, which is selected if none of the associated navigation nodes has been reached yet.

The inclusion of alternatives works just like the inclusion of presentation elements in normal presentation classes, so each alternative presentation class is used as the type of a presentation property that is owned by the presentation group.

Generalizations

- *PresentationClass* on page 13

Attributes

No additional attributes.

Associations

- default :
PresentationProperty
[0..1] Defines which presentation class is used as the default when none of the alternatives' associated navigation nodes has been reached yet.

5.1.6 UIElement

UIElement is the abstract super class for presentation elements that are responsible for presenting or editing content. Every *UIElement* has to be included in a presentation class that is associated with a navigation node. The subclasses of *UIElement* can be divided into four groups:

- *UI containers* like forms can contain other UI elements.
- Static elements, such as Image or Text, are used to display content. They can be connected with a navigation property to specify where the displayed data is retrieved from, as described in section 5.1.3. Alternatively, they can be used to provide values for query parameters.
- Elements that handle user input like TextInput or Choice. They can be connected with a navigation- or process property in order to specify how the user input is handled.
- Anchor and Button both trigger transitions on the navigation model or process model.

It is important to remember that the UWE presentation model does not specify concretely how an UI element is rendered in terms of which element of the used presentation technology is used. For example, a UWE choice that allows selection of one element could be rendered by an HTML `<select>` element as well as by a group of radio buttons.

Generalizations

- *PresentationElement* on page 13

Attributes

No additional attributes.

Associations

- `uiContainer : UIContainer [0..1]` The UIContainer that contains the UIElement.

5.1.7 UIContainer

A *UIContainer* is an abstract super class not linked to any data by itself but can include other *UI elements*.

Generalizations

- UIElement on page 15

Attributes

No additional attributes.

Associations

- `elements : UIElement [*]` The contained UI elements.

5.1.8 Form

A form groups user interface elements that are used to provide data for a process.

Generalizations

- UIContainer on page 16

Attributes

No additional attributes.

Associations

No additional associations.

5.1.9 AnchoredCollection

An anchored collection is an *UI container* that can only contain anchors. It can be used to model the presentation of a menu or an index.

Generalizations

- UIContainer on page 16

Attributes

No additional attributes.

Associations

- `anchors : Anchor [1..*]` The anchors contained by the anchored collection.

5.1.10 Anchor

An anchor allows the user to trigger a transition in the navigation model alongside a specified link. Note that the UWE presentation model does not specify how an anchor is rendered. In HTML, for example, both an anchor element (`<a>`) as well as a button may be used.

Generalizations

- UIElement on page 15

Attributes

No additional attributes

Associations

- link : Link [0..1] The link that is followed when the anchor is clicked.

5.1.11 Button

A button in general is an element that enables the user to initiate some action of the web application. The most common usage is in conjunction with input elements to submit data and execute a query or a process. Just like mentioned in section 5.1.10, UWE does not specify how a button is rendered. If HTML is used as presentation technology, an `<input>` element with type "button" could be used as well as an `<a>` element or even an image, (given that JavaScript is enabled).

Generalizations

- UIElement on page 15

Attributes

No additional attributes

Associations

No additional associations.

5.1.12 Text

A text element is used to displays static text. The content can be provided by a navigation property as described in section 5.1.3.

Generalizations

- UIElement on page 15

Attributes

No additional attributes

Associations

No additional associations.

5.1.13 Image

An image element is used to display a static image. The content provided by the corresponding navigation property (see section 5.1.3) could be interpreted as an URL specifying the location of an image file or directly as image data in any format.

Generalizations

- UIElement on page 15

Attributes

No additional attributes

Associations

No additional associations.

5.1.14 TextInput

A text input element allows the user to enter text.

Generalizations

- UIElement on page 15

Attributes

No additional attributes

Associations

No additional associations.

5.1.15 Choice

A choice allows selecting one or more values from a set of possibilities. In a web application, there are several different ways how this functionality could be realized by concrete HTML elements, e.g.:

- By a <select> element
- By a group of radio buttons to select one value out of several values
- By a group of checkboxes to select multiple values
- By one checkbox if the edited property is of type Boolean

Generalizations

- UIElement on page 15

Attributes

- multiple : Boolean
(default = false) Defines whether the choice allows selecting more than one value.

Associations

No additional associations.

6 Process Package

The process package provides model elements for integrating business processes into an UWE web application model. This can be separated into three tasks:

- **Integration of business processes into the navigation model**

This is enabled by the two metaclasses *ProcessClass* and *ProcessLink* that extend *Node* and *Link* respectively and that allow defining how a process can be reached through navigation and how navigation will continue after the process.
- **Definition of a user interface to support the processes**

Processes most likely require a user interface for data input and presentation. This user interface can be defined with the UWE presentation model for each process class just like the UI for navigation classes as described in section 5. However, user input may be required at several points in the process flow. This is solved by creating one process class for each step and associating them with the main process class that is integrated in the navigation model. For each of these process classes, a presentation class will be created defining the user interface. The UI elements are connected with process properties of the corresponding process class.
- **Definition of the behaviour**

The behaviour of a process is defined by an UML activity that is owned by the main process class. The following restrictions and special semantics apply:

 - A special *UserAction* is used to mark a point in the control flow when the user is asked to enter data. The *user action* is associated with a *process class* to identify what data is edited and what *presentation class* is shown. The control flow of the activity continues after the user has submitted the requested data. Each *process property* of the *process class* provides entered data from the corresponding *UI element* through an output pin of the *user action* that has the same name as the *process property*. Similarly, the *process properties* of a *process class* can be set with input pins of the corresponding *user action*. These values are used as initial values for the connected *UI elements*.
 - In many cases, a process needs some input from its predecessor node in the navigation graph. For example, an `EditContact` process would need an instance of the content class `Contact` as input. This instance could be provided by a navigation class `Contact` from which the user, over a menu, can chooses to edit the particular contact. This situation can be modelled by an activity parameter node that is used instead of an initial action node. The parameter node must have the same type as the content class of the navigation class that precedes the process class.
 - The actions in the process activity that are not *user actions* may call operations of the input parameter object and on every instance that is created during the process activity. How access to other contexts is expressed is up to the modeller.
 - The process could create or select a content class instance that should be passed to a succeeding node (navigation class or process class). This can be modelled by an activity parameter node that is used instead of an activity final node.
 - Other processes can be embedded by calling the corresponding process activity using UML *CallBehaviorActions*.

The model elements presented above and the relationships between them are shown in Figure 6. These model elements are described in the following subsections.

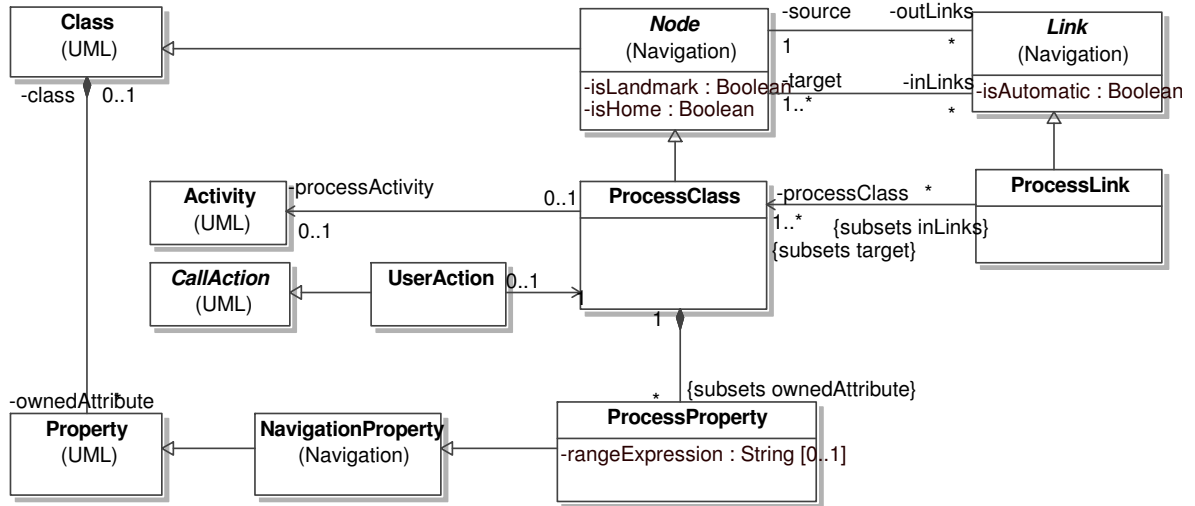


Figure 6: The Process Package

6.1 Class Descriptions

6.1.1 ProcessClass

Process classes are used to integrate business processes into the navigation model and to define the data that is exchanged with the user during the process.

In the navigation model, process classes can be connected to other navigation nodes using process links. This defines how a process can be reached through navigation. If a process involves several steps with different user interfaces, each step has to be backed up by a process class that is associated with a user action (see section 6.1.4). The user interface of each step is defined by a presentation class that is associated with the process class using the “node” role. However, only one class is integrated in the navigation model. This class becomes the “main process class” and has to be associated with the activity that defines the process flow.

The properties of *process classes* (process properties) are each connected with a *UI element* and provide means to define how data retrieved from the user interface is used within the process (see sections 6.1.3 and 6.1.4).

Generalizations

- Node on page 7

Attributes

No additional attributes

Associations

- processActivity : Activity [0..1] The UML activity that defines the process flow. This is only used for the main process class of a business process (the one that's used as a node in the navigation model). As an abbreviation, the activity can be encapsulated in the process class using the `ownedBehavior` feature.
- processProperty : ProcessProperty [*] {subsets ownedAttribute} A collection of properties that are each connected to a UI element and are used to define how input from these UI elements is handled by the process.

6.1.2 ProcessLink

Process links are used to connect process classes to other navigation nodes.

Generalizations

- Link on page 7

Attributes

No additional attributes.

Associations

- processClass : ProcessClass [1..*] The target node(s) of the process link.

6.1.3 ProcessProperty

A *process property* is owned by a process and is used to define how data retrieved from a *UI element* is used within the process flow. The relation to the *UI element* is established by the feature `navigationProperty` of *PresentationProperty* (see section 5.1.3).

Generalizations

- NavigationProperty on page 8

Attributes

- rangeExpression : String [0..1] An expression that can be used to define a range of possible values for input into the related *UI element*.

Associations

No additional associations.

6.1.4 UserAction

A *user action* defines a point in the process flow when the user is asked to input data. It is associated to a process class that in turn is referenced by a presentation class. When the user action is reached in the control flow of the process activity, the *UI elements* of the corresponding presentation class are shown. After the user has submitted data, the process flow is continued. The data that has been entered in the user interface elements is available via

output pins of the user action that are named equal to the process properties that back up the *UI elements* (see section 6.1.3). Analogically, input pins can be used to define that data from the activity’s object flow should be displayed by the corresponding UI elements.

Generalizations

- *CallAction* (from UML)

Attributes

No additional attributes.

Associations

- processClass : ProcessClass [1] A process class that is referenced by a presentation class and that provides process properties that are on their part referenced by UI elements.

7 UWE Profile

The UWE metamodel is mapped to a UML profile. The definition of a UML profile has the advantage that it is supported by nearly every UML CASE tool. The semantics of the stereotypes corresponds to the elements of the same name in the metamodel.

UWE stereotype	UML base class	Used in	Icon
«anchor»	class	presentation model	—
«anchored collection»	class	presentation model	☰
«button»	class	presentation model	●
«choice»	class	presentation model	
«form»	class	presentation model	☐☐
«guided tour»	class	navigation model	➤
«image»	class	presentation model	●☐
«index»	class	navigation model	☰
«menu»	class	navigation model	☐☐☐
«navigation class»	class	navigation model	☐
«navigation link»	association	navigation model	
«navigation property»	property	navigation model	
«page»	class	presentation model	📄
«presentation class»	class	presentation model	☐○
«presentation group»	class	presentation model	





UWE stereotype	UML base class	Used in	Icon
«presentation property»	property	presentation model	
«process class»	class	navigation/process model	
«process link»	association	navigation model	
«process property»	property	navigation/process model	
«query»	class	navigation model	
«text input»	class	presentation model	
«text»	class	presentation model	
«user action»	action	process model	

Table 1: UWE Stereotypes

8 Example: Simple Music Portal

The following example is meant to illustrate as many of the issues discussed in this document as possible. It models a very simple music portal web application that allows users to buy albums which then can be downloaded as archive files containing MP3s. The objective of the simplified example is to show how to use all UWE model elements when building models of web applications. The following list gives a short informal description of the use cases and requirements.

- A distinction is made between users and registered users. Only registered users can buy or download albums. A user becomes a registered user by logging in. Unregistered users can register with a username that has not been taken by another user and a freely chosen password.
- Every user can search for albums by their name. Other search methods are not offered. The search result is presented as a list of matching albums that provides links to a detail page for each album. The album detail pages show the title of the album, the name of the artist, the list of songs and the album's price. If the user has already bought the album then a download link is shown. Otherwise, there will be a link for buying the album.
- Only full albums can be downloaded.
- In this simplified example, each album has only one artist. This restriction is done to reduce the complexity of the navigation and presentation models. It would be easy though to add support for multiple artists by adding an index in the navigation model and an anchored collection in the presentation model, respectively.
- Each registered user has a credit account that is used to buy albums. The credit account can be recharged by credit card payment. To do this the user has to enter her credit card data and the amount to recharge with. This data is validated and the user has to confirm the transaction before the credit card is charged and the user's credit account is recharged. The details of credit card handling are not modelled in this example.
- If a user is logged in, she can navigate to an account page that shows the user's credits and the list of albums she has bought in the past.
- The links for logging in or out, for registering and to the user's account page are always shown. This also holds for the album search box.

The example uses a shorthand notation that omits tagged values and uses name-matching to establish the relationship between model elements. For example, the navigation class "Album" should actually have a tagged value "contentClass=content::Album", but as the names for content and navigation class are equal, the tagged value can be left out. The same pattern is used for the connections between presentation classes and navigation nodes and between presentation and navigation properties. Also, the stereotype «navigationLink» is not shown.

8.1 Use Cases

Figure 7 shows the use case model of the music portal Web application. Two actors are triggering the use cases: the (anonymous) user and the registered user. The web application supports non-transaction-based functionality, such as searching and viewing of albums and songs as well as transactional functionality, such as recharging and downloading an album.

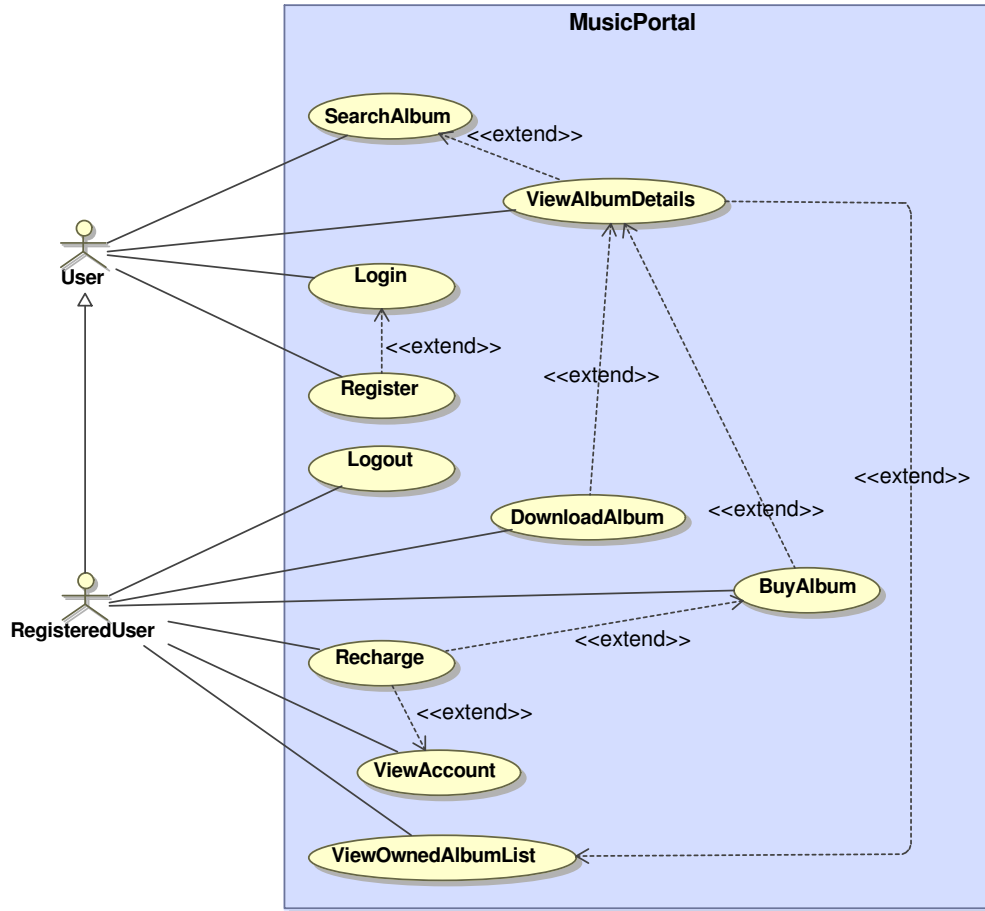


Figure 7: Music Portal Use Cases

8.2 Content Model

The content model visualizes the domain relevant information for the Web system that mainly comprises the content of the Web application. In our example the information is provided by the classes `Album`, `Artist` and `Song`. A UML class diagram and UML plain classes are used to model the content.

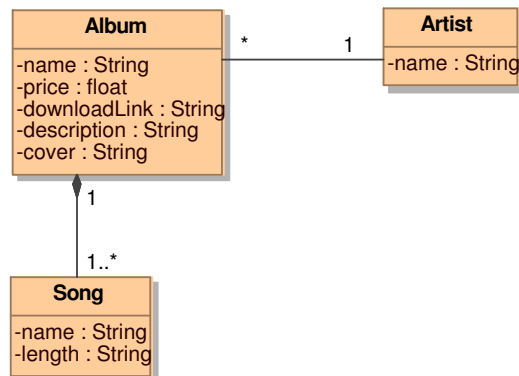


Figure 8: Music Portal Content Model

8.3 User Model

The separation in user model and content model is a decision that is up to the modeller. Only plain UML elements are used as no additional semantics applies. While the content model defines the data content of the application, the user model serves two different purposes. On the one hand, it contains classes that define what information is stored in the context of a session. In this example, we see that a session can have one current user who can have a collection of owned albums. On the other hand, the classes in the user model provide operations that can be used in the business processes. The behaviour of these operations is not modelled but has to be implemented separately. For example, the operation `CreditCard::charge()` will probably be realised by calling a third party library or web service. For some of the other methods, the behaviour is described using OCL. This is just one possible method though, as UWE does not specify how to define operation behaviour.

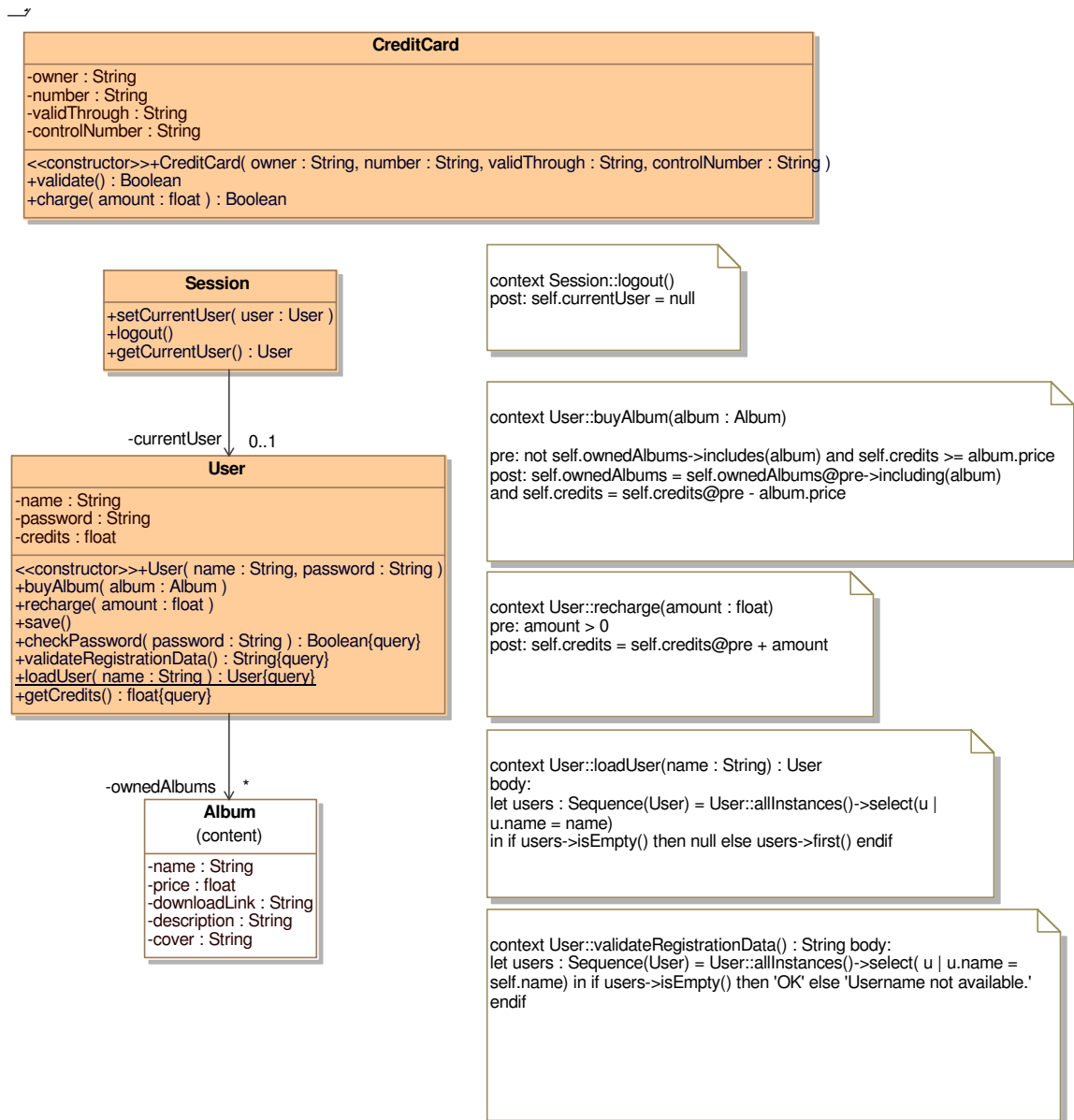


Figure 9: Music Portal User Model

8.4 Navigation Model

The navigation model of the example is designed as described in section 4. The navigation is strongly simplified and is mainly intended to demonstrate the use of the model elements than to show a realistic example.

As mentioned above, the stereotype «navigationLink» is not shown on associations to make the diagram more readable. Also the content classes of the navigation classes are not specified explicitly but are rather derived by name matching with classes from the content model. Only the one navigation attribute that needs a selection expression is defined explicitly (Album::artistName). The others are derived implicitly from the properties of the content classes.

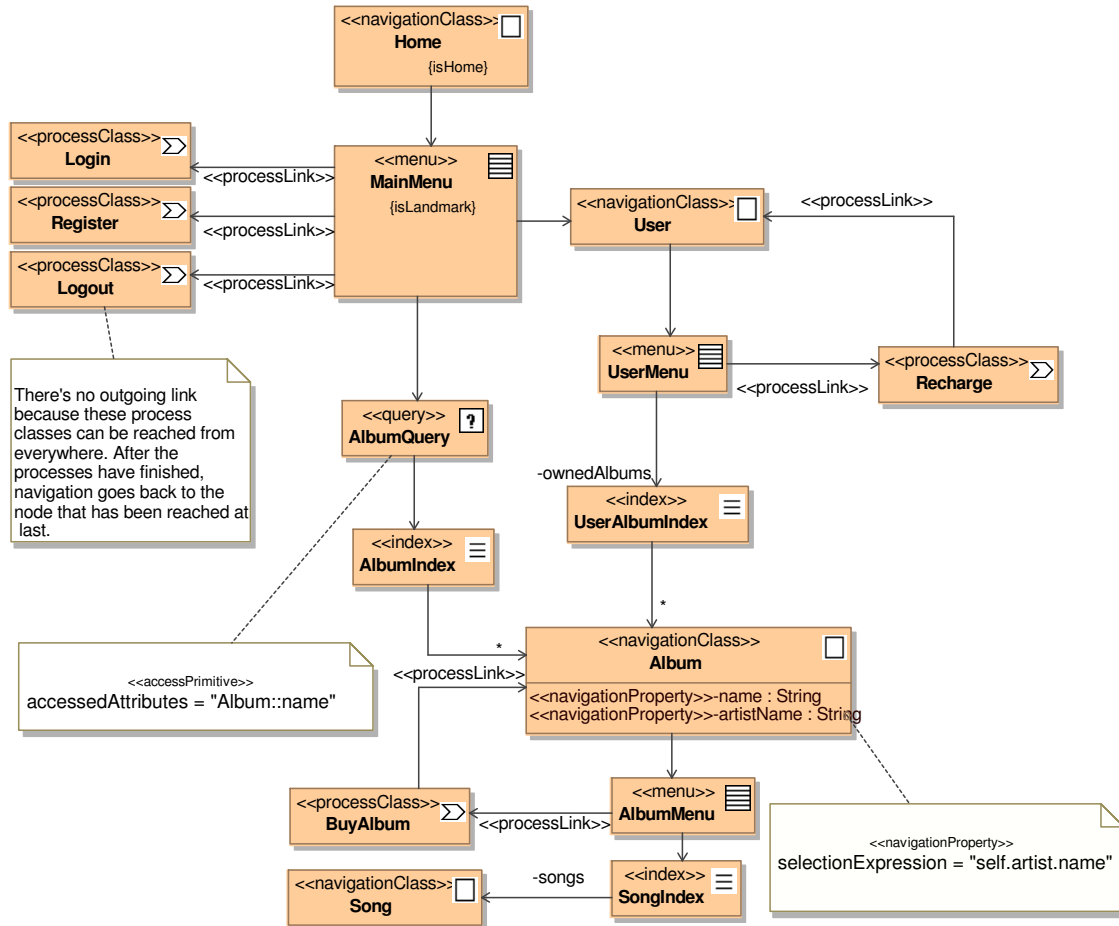


Figure 10: Music Portal Navigation Model

8.5 Business Processes

The example's process model consists of the process classes that are integrated in the navigation model, additional process classes to handle user input and activities that define the behaviour of the processes (see section 6). The process model of Figure 11 shows the relationship of main process class and process classes, which have associated the process flow.

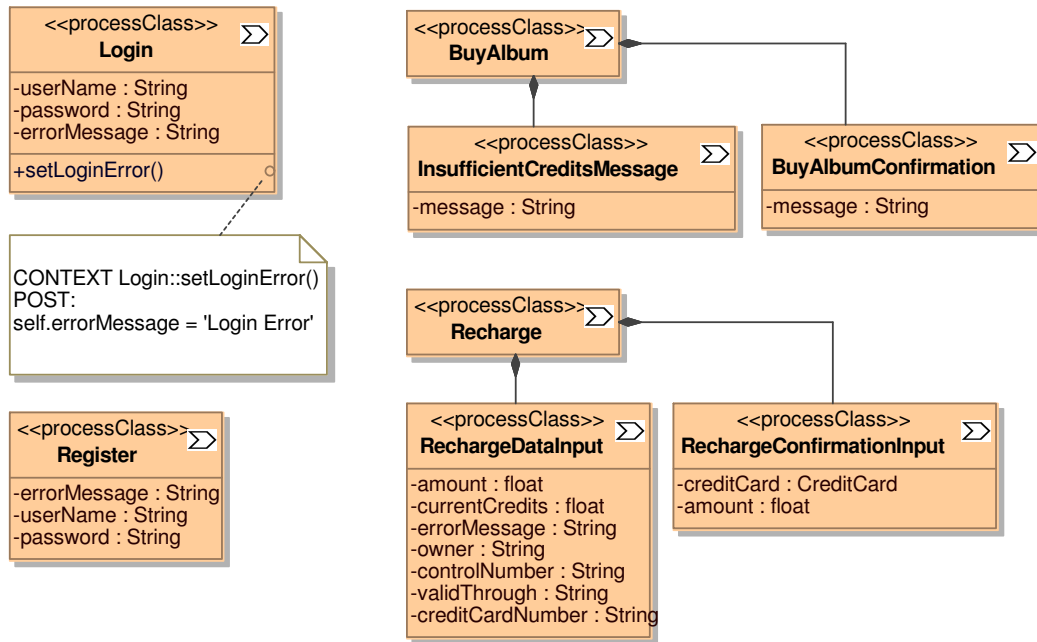


Figure 11: Music Portal Structural Process Model

The workflows modelling the behaviour of the process classes Login, Register, BuyAlbum and Recharge are shown in Figure 12 to Figure 16.

8.5.1 Process Login

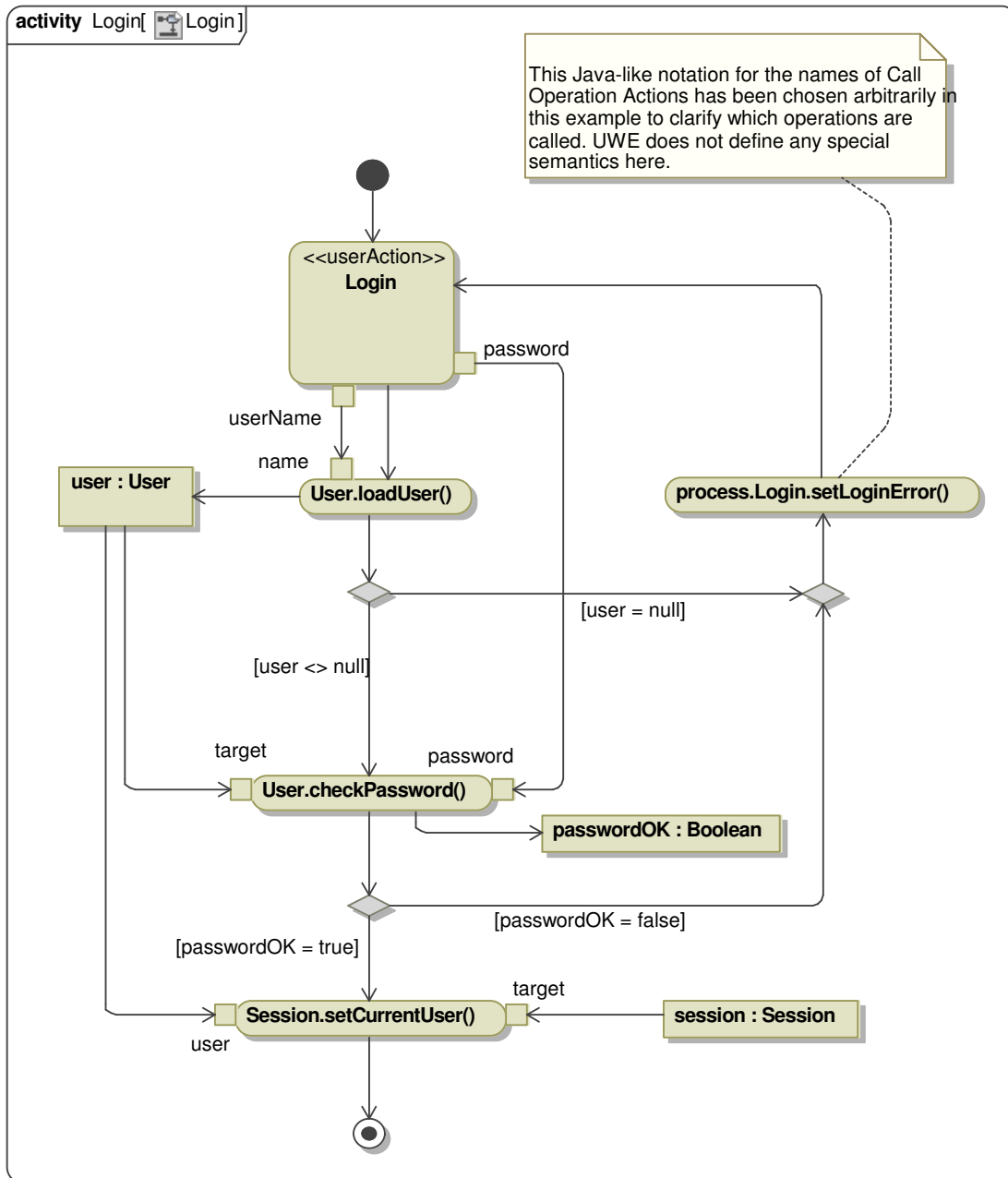


Figure 12: UML Activity Diagram for the Process Login

8.5.2 Process Logout

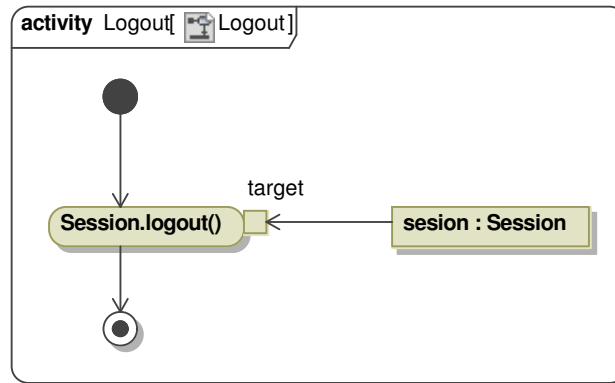


Figure 13 UML Activity Diagram for the Process Logout

8.5.3 Process BuyAlbum

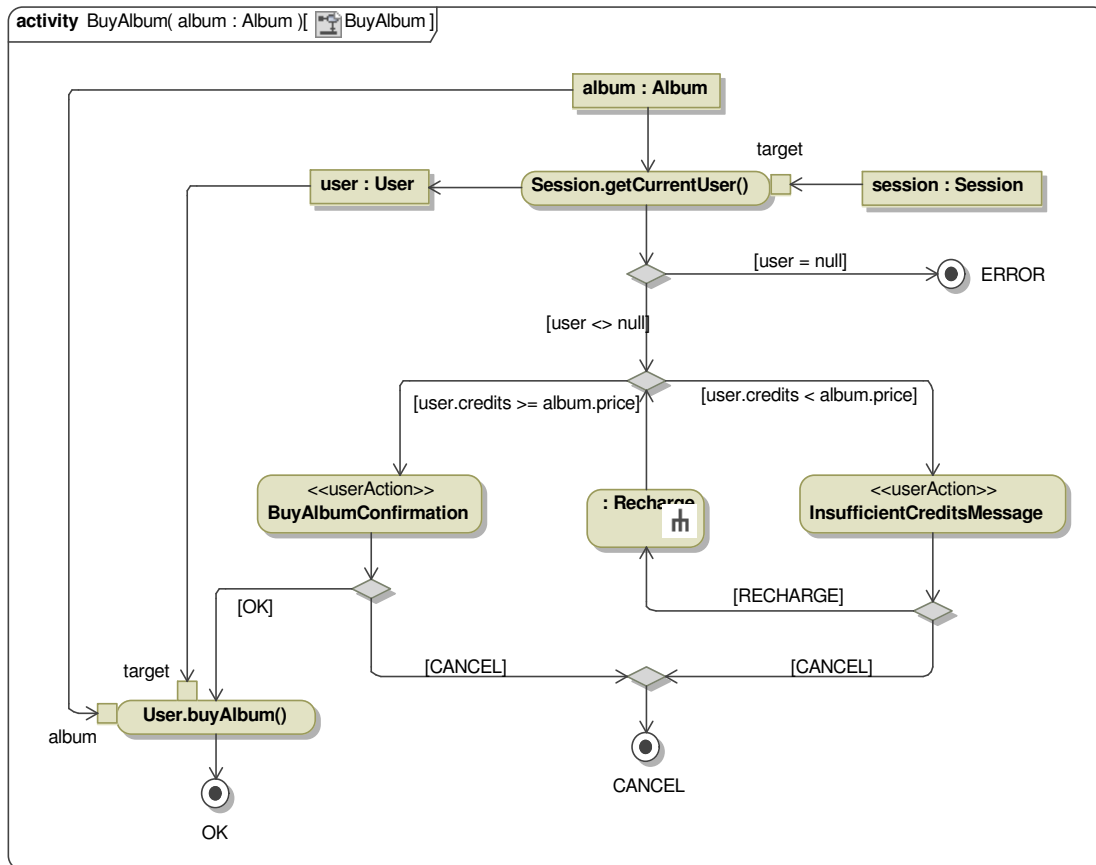


Figure 14: UML Activity Diagram for the Process BuyAlbum

8.5.4 Process Register

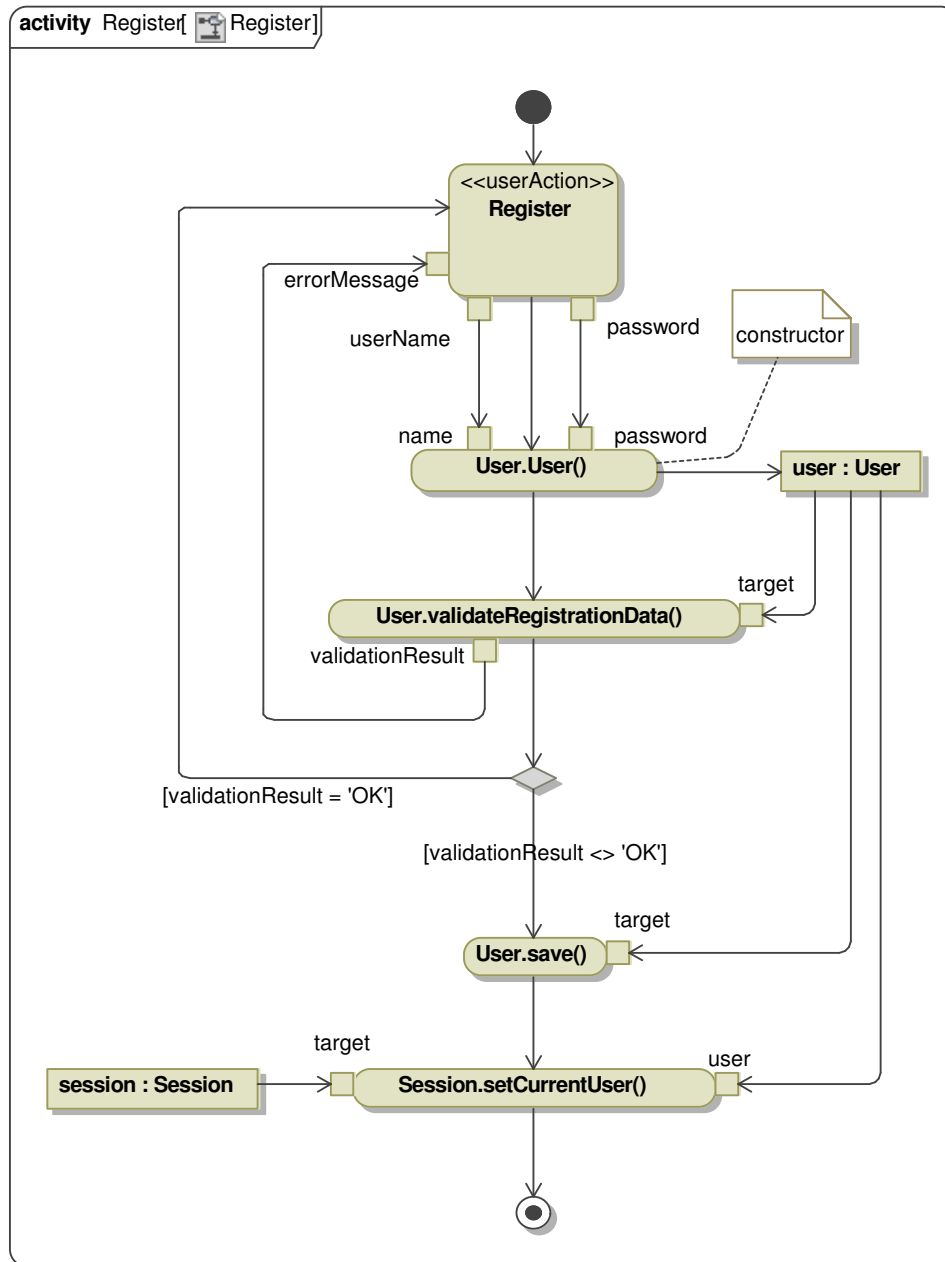


Figure 15: UML Activity Diagram for the Process Register

8.5.5 Process Recharge

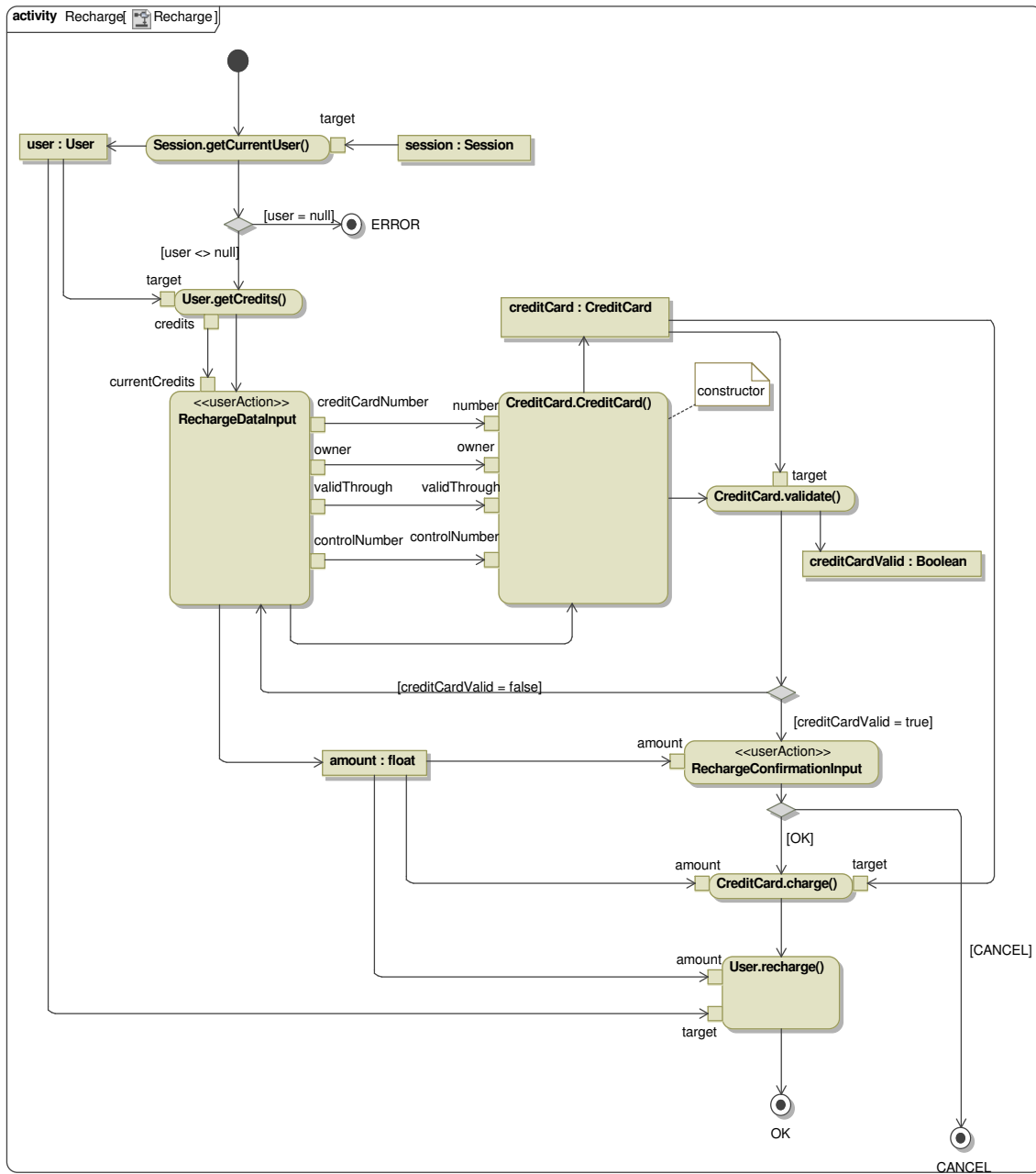
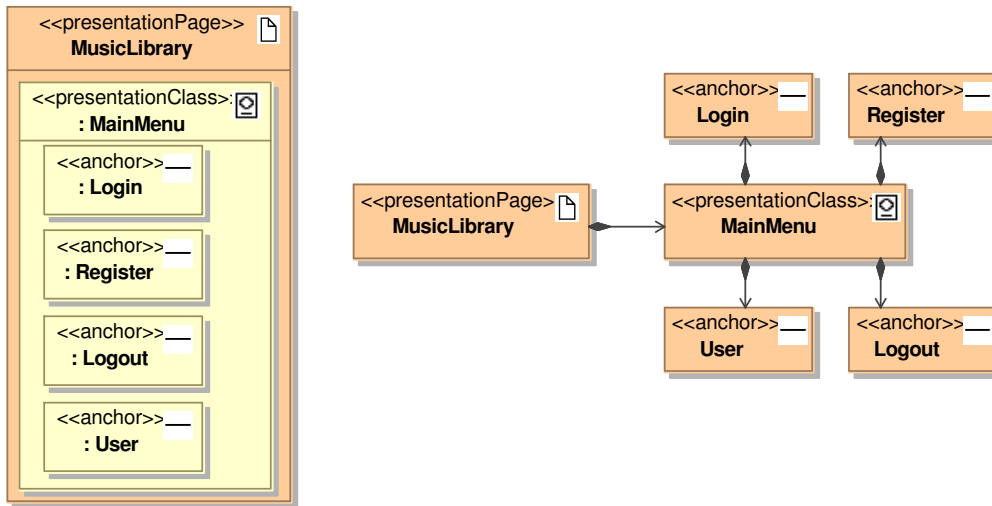


Figure 16: UML Activity Diagram for the Process Recharge

8.6 Presentation Model

The presentation model of the example is shown as a UML composite structure diagram. In this kind of diagram, properties that are contained by composition are shown as rectangles that are contained in the figure of the containing class. For example the presentation class `MainMenu` could be shown in the following two equivalent ways:



Note: The contained figures represent properties not instances.

The shorthand notation described above, with name matching instead of tagged values, is used intensively for presentation model diagrams:

- The navigation nodes of presentation classes are chosen by name.
- UI elements are implicitly connected with equally named navigation properties (e.g. the «text» element `title` in the «presentationClass» `Book`). Note that as described in section 4.1.4, the navigation properties could be derived implicitly from the content model as well.
- The link that is followed when an anchor is clicked is determined by matching the anchor's name with target role names of links that originate from the navigation node that is connected to the presentation class that contains the anchor. If there's only one link to a particular navigation node, the node's name can be used as an implicit role name. For example, the anchors `Login`, `Register`, `Logout` and `User` are implicitly related to the links from `MainMenu` to the respective process or navigation classes.

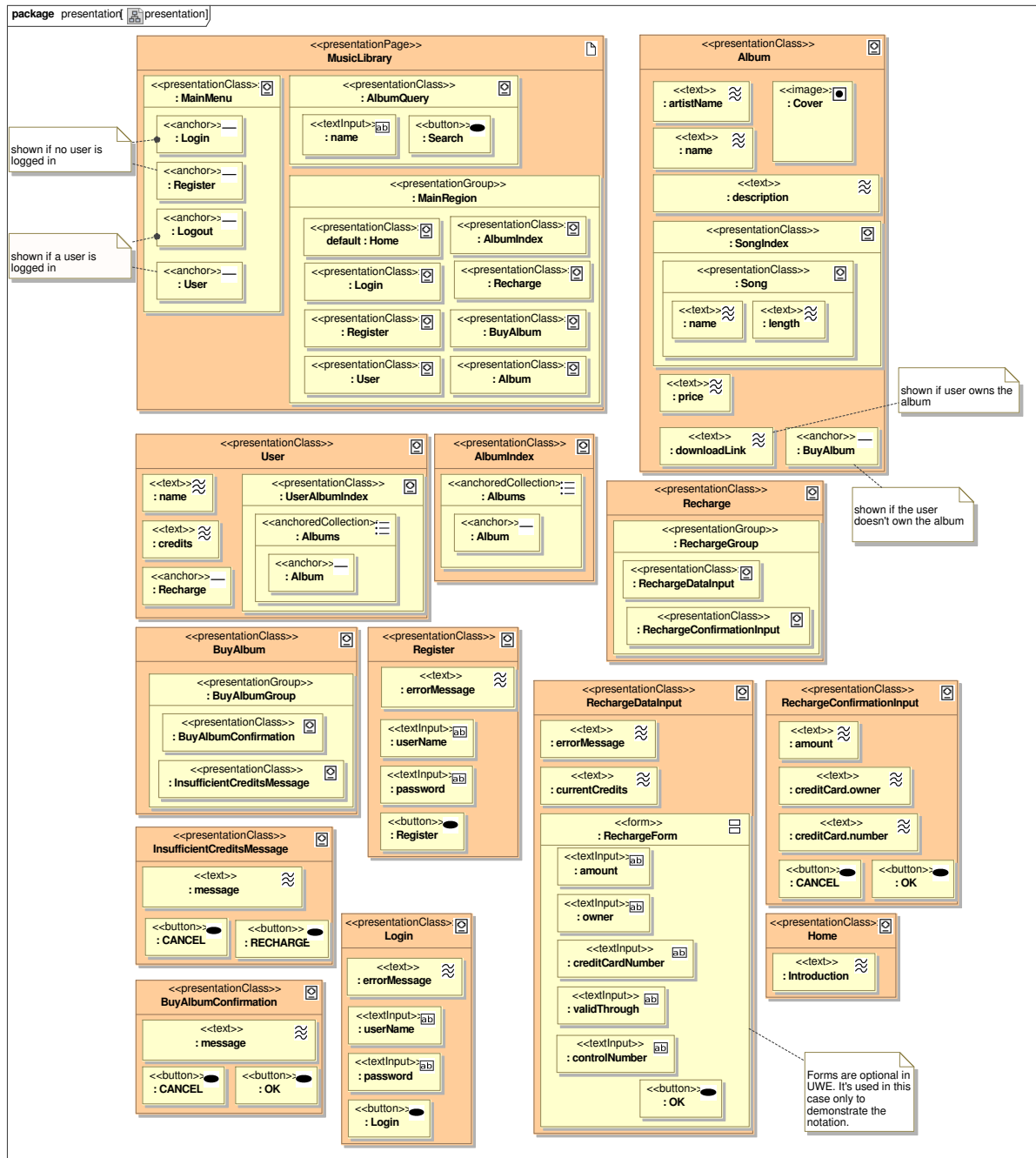


Figure 17: Music Portal Presentation Model

References

- [1] Nora Koch, Alexander Knapp, Gefei Zhang and Hubert Baumeister. UML-based Web Engineering: An Approach based on Standards (book chapter). In *Web Engineering: Modelling and Implementing Web Applications*. Gustavo Rossi, Oscar Pastor, Daniel Schwabe and Luis Olsina (Eds.), Springer, HCI, November 2007.
- [2] Nora Koch and Andreas Kraus. Towards a Common Metamodel for the Development of Web Applications. In Juan Manuel Cueva Lovelle, Bernardo Martín González Rodríguez, Luis Joyanes Aguilar, José Emilio Labra Gayo, and María del Puerto Paule Ruíz, editors, *Proc. 3rd Int. Conf. Web Engineering (ICWE 2003)*, volume 2722 of LNCS, pages 497-506. Springer Verlag, 2003.
- [3] Andreas Kraus and Nora Koch. A Metamodel for UWE. Technical Report 0301, Ludwig-Maximilians-Universität München, 20 pages, January 2003.