Vrije Universiteit Brussel

FACULTY OF SCIENCE AND BIO-ENGINEERING SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

Integrating the Interaction Flow Modelling Language (IFML) into the Web Semantics Design Method (WSDM)

Master thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

Jonas Blanckaert

Promoter: Prof. Dr. Olga De Troyer
Advisor: Pejman Sajjadi

Academic year 2014-2015

Vrije Universiteit Brussel

FACULTEIT WETENSCHAPPEN EN BIO-INGENIEURSWETENSCHAPPEN
VAKGROEP COMPUTERWETENSCHAPPEN

Integratie van de Interaction Flow Modelling Language (IFML) in de Web Semantics Design Method (WSDM)

Masterproef ingediend in gedeeltelijke vervulling van de eisen voor het behalen van de graad Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

Jonas Blanckaert

Promotor: Prof. Dr. Olga De Troyer
Begeleider: Pejman Sajjadi

Academiejaar 2014-2015

Abstract

Web development for large web applications inherently requires an appropriate development method. The Web Semantics Design Method (WSDM) is such a development method. It focuses on the different audiences that will visit the web system. WSDM consists of different design and modelling phases and sub-phases, each with its own modelling language. Some of these languages are WSDM-specific and not generally known. It would be beneficial to WSDM to have these languages replaced with other well-known standardised languages. The high amount of modelling languages in WSDM might affect the amount of time required to learn and understand WSDM for a developer. If the number of languages used could be decreased, the learning curve of WSDM would be less steep.

The Interaction Flow Modelling Language (IFML) is an Object Management Group (OMG) standard which facilitates interaction design of applications. This thesis investigates whether IFML could be integrated into WSDM to meet its standardisation and simplification needs. To conduct this investigation, the objectives of each step in WSDM are compared with those of IFML to evaluate exactly where IFML would fit into WSDM. We found that IFML can be integrated into WSDM. More specifically, in the Navigational Design and Site Structure Design sub-phases. However, this integration required the definition of a conceptual version of IFML, to be used in the Conceptual Design phase. We propose an adapted version of WSDM that adopts this integration. Finally, we conclude that, due to the specific goals of the modelling languages in WSDM, it is infeasible to replace more than two modelling languages (the navigational modelling language and its site structure design extension) with IFML. However, the standardisation needs of WSDM are fully met. Moreover, we propose an algorithm to automatically carry out the Navigational Design sub-phase.

Samenvatting

Web ontwikkeling van grote websites vereist onvermijdelijk een geschikte ontwikkelingsmethode. De Web Semantics Design Method (WSDM) is zo een methode. In deze methode staan de behoeften van de verschillende soorten gebruikers die de website zullen bezoeken centraal. WSDM bestaat uit een aantal design- en modelleerfasen en sub-fasen die elk een eigen modelleertaal gebruiken. Sommige van deze talen zijn eigen aan WSDM en daarbuiten niet bekend. Het zou voordelig zijn voor WSDM om deze onbekende modelleertalen te vervangen door andere, beter bekende en gestandaardiseerde talen. Het hoge aantal modelleertalen in WSDM leidt ertoe dat de tijd die een ontwikkelaar nodig heeft om WSDM te leren groot is. Als het aantal modelleertalen verkleind zou worden, zou dit de leercurve van WSDM kunnen verkleinen.

De Interaction Flow Modelling Language (IFML) is een Object Management Group (OMG) standaard die het design van interacties in applicaties ondersteunt. Deze thesis onderzoekt of IFML geïntegreerd zou kunnen worden in WSDM om aan de standaardisatie- en vereenvoudigingsnoden van WSDM te voldoen. Om dit onderzoek uit te voeren, worden de doelen van de diverse stappen in WSDM vergeleken met deze van IFML om exact te kunnen bepalen waar IFML in WSDM zou kunnen passen. Ons onderzoek wees uit dat IFML inderdaad kan geïntegreerd worden in WSDM, namelijk in de Navigational Design sub-fase en de Site Structure Design sub-fase. Deze integratie gaat echter gepaard met de definitie van een conceptuele versie van IFML voor gebruik in de Conceptual Design fase. We presenteren de aangepaste versie van WSDM die deze integratie bevat. Ten laatste kunnen we concluderen dat, omwille van de specifieke doelen van de modelleertalen in WSDM, het niet haalbaar is om meer dan twee modelleertalen (de navigatie modelleertaal en haar website structuur design extensie) te vervangen door IFML. Hoe dan ook is de standaardisatie nood van WSDM ingevuld door de aangepaste WSDM versie. Bovendien presenteren we een algoritme dat de Navigational Design sub-fase volledig automatiseert.

Declaration of Originality

I hereby declare that this thesis was entirely my own work and that any additional sources of information have been duly cited.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this thesis has not been submitted for a higher degree to any other University or Institution.

Acknowledgements

To my promotor Prof. Dr. Olga De Troyer and advisor Pejman Sajjadi: I would like to thank you for your continuous support, availability and guidance throughout the completion of this thesis. I greatly appreciate the time you made for our insightful discussions.

— Jonas

Contents

List of Figures

1 Introduction

1.1 Context

When developing large or complex web systems it is crucial to guide the development process by an appropriate method. Such a method defines a starting point and a subsequent stepped and/or iterative process that keeps the developer on the right track. It seeks to structure and select its phases as to satisfy a set of business concerns, such as minimal time and cost to develop or/and a high quality and maintainability of the system, but also other concerns, such as an appropriate level of complexity (i.e., the developer should not get lost in an overly complex method). Such a method should abstract from these concerns, in order to prevent that the developer constantly has to question whether what he is doing is the right thing to do at that time in the development process. Instead, he should be able to trust that the method will output a satisfactory solution.

A range of web development and design methods has been subject of research in the past decades, some of which have radically different views on the development process. While certain methods use a data or object-driven approach, others centre the development on the needs of the users or the audiences of the web system under construction. Nevertheless, most methods have phases and order them conform a common general pattern: first conceptual modelling and navigational modelling is conducted, then

presentation design, and finally implementation.

The Web Semantics Design Method (WSDM) (De Troyer et al., 2008) is a method which uses an audience-centred approach. It starts by determining which audiences will visit the website and which their requirements are. From those requirements, it constructs task models which describe the tasks that these audiences can perform with the system. Then the information needed to carry out those tasks and the navigational links between them are modelled. At that stage, the specified models in WSDM are purely conceptual. In the next step, the conceptual system is transformed into a concrete implementation-specific model through specifying exactly how the user interface of the system will look and how the used data is mapped to a specific data source. Finally, the web system is implemented using the models obtained.

The Interaction Flow Modelling Language (IFML) (Object Management Group, 2015) is a standard specified by the Object Management Group (OMG)[1]. It is used to model the interactions between the user and the front-end of an application. The first version of IFML (IFML 1.0) has only recently (2015) been officially published.

IFML has strong expressive qualities. Whilst it captures the different user interface components and the actions that can be carried out with them in detail, it also structures the interface components. An IFML model expresses the components that reside in the same interface window in the application and which of them are visible at the same time. Its visual appearance allows the application developer to understand the general structure of the application without in-depth analysis of the model.

1.2 Research Objective

WSDM has many different phases and sub-phases. Each of these phases has its own specific technique, modelling language and notation to complete its objective. The advantage of using specialized languages is that the output of the phases is always presented in a language that is particularly well-suited to present the output of this type. For instance, a graphical modelling language allows to easily spot certain parts of the model, or to abstract away from unnecessary details.

However, as there are many different modelling languages used in the WSDM phases[2], it takes a lot of time to study and learn WSDM thoroughly.

[1] http://www.omg.org/

[2] For example, ConcurTaskTrees (CTT), Object-Role Modelling (ORM), the naviga-

It also becomes complex to reason about the outputs of the different phases, as it is needed to mentally switch between modelling languages. It would hence be advantageous if the amount of modelling languages in WSDM could be reduced in such a way that the web developer is not required to be an expert in a large set of different modelling languages.

WSDM uses the CTT and ORM notations which are generally accepted in the conceptual modelling community. The other notations used in WSDM, such as the navigational modelling language and its site structure design extension, are WSDM-specific. It is desirable to replace the WSDM-specific languages as well as the generally accepted (non-standardised) languages by other languages which are standardised, in order to make WSDM more widely accepted and supported.

The research objective of this thesis is to investigate whether IFML can be integrated in WSDM in order to achieve the simplification and standardisation needs expressed above.

1.3 Research Method

The research methodology used to conduct the research in this thesis is a variant of the Design & Development-centred approach of the Design Science research methodology presented in Peffers, Tuunanen, Rothenberger, and Chatterjee (2007). The following paragraphs describe the steps that led to the proposed adapted WSDM which is the result of this thesis.

Problem Identification and Objectives for a Solution

As elaborated upon in the previous section, WSDM would benefit from a simplification in terms of the number of used modelling languages. Also would it be advantageous to use standardised modelling languages in the method in order to improve acceptance and accessibility of WSDM. The ideal language to integrate in WSDM is therefore a standardised language which has a wide coverage. A possible solution was seen in the integration of IFML in WSDM, as it is a promising standardised language that could cover specific sub-phases of WSDM to some extent.

Design and Development

This step is the entry point of this thesis in the methodology. This thesis has as objective to evaluate how IFML could fit into WSDM. Thereto, each

tional modelling language, and the site structure design modelling language.

phase and sub-phase of WSDM is considered and has its properties compared with those of IFML in order to assess whether there is overlapping and an opportunity for integration. Along with this assessment process, an integration is proposed which we believe to be the most feasible one in terms of the desired properties of WSDM, specified in the research objective.

Demonstration

At last, a case study is presented which demonstrates the capabilities of the solution developed in the previous step: the adapted WSDM.

1.4 Structure of the Thesis

This thesis document is structured as follows. In Chapter 2, the reader is presented an overview of both IFML and WSDM, to be able to thoroughly understand the discussions in subsequent chapters, would the reader not yet be acquainted with IFML or WSDM. Chapter 3 mentions other work which is closely related to the subject of this thesis. Chapter 4 comprises the main contribution of the thesis as it seeks to find a concrete way to integrate IFML in WSDM. Thereafter, Chapter 5 discusses a large case study as a proof-of-concept of the adapted WSDM which was constructed in the previous chapter. Finally, a conclusion of this work is given in Chapter 6.

2 Background

This thesis contributes to integrating the *Interaction Flow Modelling Language* (IFML) in the *Web Semantics Design Method* (WSDM). Therefore, this chapter will start by providing the reader an overview on both IFML and WSDM.

2.1 Interaction Flow Modelling Language (IFML)

The *Interaction Flow Modelling Language* (IFML) is a language specified by the *Object Management Group* (OMG) to describe *interaction flow models*. These models are used to describe front-end user-application interaction. A description of the purpose of these models and how they can be constructed using IFML will be the topics of the next subsections. In the last two subsections, examples of IFML diagrams will be shown and analysed to provide a more concrete understanding of IFML, and some remarks on the semantics of IFML will be made.

At the time of writing, the most recent IFML specification (Object Management Group, 2015) specifies the language's first official version (IFML 1.0) (released February 2015). It is available to download at the OMG website[1].

[1] http://www.omg.org/spec/IFML/1.0/

2.1.1 Introduction to Interaction Flow Models

Certain software applications can remain in use over large time spans. Think of several years or possibly decades. As the world around them evolves, creating a continuous flow of changing requirements, these applications need to stay tuned. This often involves making major changes to the programs. Think of today's rapidly changing security requirements. These might cause businesses to be forced to change the security technology which supports their programs, to stay up-to-date and protected. Changing software usually comes along with a great cost (Beck, 1999). However, the productivity can be increased —and so the cost reduced— if applications are developed using a *model-driven architecture*[2] (MDA) (Kleppe, Warmer, & Bast, 2003). The MDA approach places the model which describes the computer program in the center of the development process, rather than the source code of the program (Kleppe et al., 2003). The model is a so-called *platform-independent model* (PIM). PIMs are not coupled to any technology or programming language involved in the implementation of the computer program. Instead, they can be transformed, using model-transformations, into one or several *platform-specific models* (PSM) which are bound to the implementation of the application. This PSM then allows the developer to directly generate the source code. When changes need to be made to a program, supported by a model-driven architecture, only the PIM or the PIM-to-PSM transformations need to be revised. Because the developer possesses a PIM, which already describes the program, changing the supporting technology merely consists of adjusting the model-transformation and regenerating the source code. This causes the mentioned productivity benefit. An example of such a platform-independent model is an interaction flow model.

Interaction flow models are PIM-level models which can be used to describe the interactions between the users of an application and the application itself. They present the user interface elements needed at the front-end of the application, without specifying layout details of these elements. This allows for separation of concerns among the developers. The user interface designer and the interaction designer can work separately. The user interface designer builds the user interface according to the interaction flow model. The model is however not limited to specifying the elements of the user interface. It also includes how data flows between the different interface elements upon triggering events and it specifies which business logic actions should be carried out using this data —without mentioning how these actions will proceed.

[2] http://www.omg.org/mda/

2.1.2 The IFML Model

As previously mentioned, interaction flow models are described using the Interaction Flow Modelling Language. Together with the *domain model* and *view points*, interaction flow models form an *IFML model*.

A domain model is a UML class diagram which describes the content available to the interaction flow model. Inside the interaction flow model, the elements of the domain model can be referenced. An example of a domain model, used for an item bidding website, is given in Figure 2.1.

View points are selections of elements of the interaction flow model that compose one specific functional aspect of the application. They can be used to isolate parts of a complex application to understand it better or to grant users access to specific parts of the application (Object Management Group, 2015).

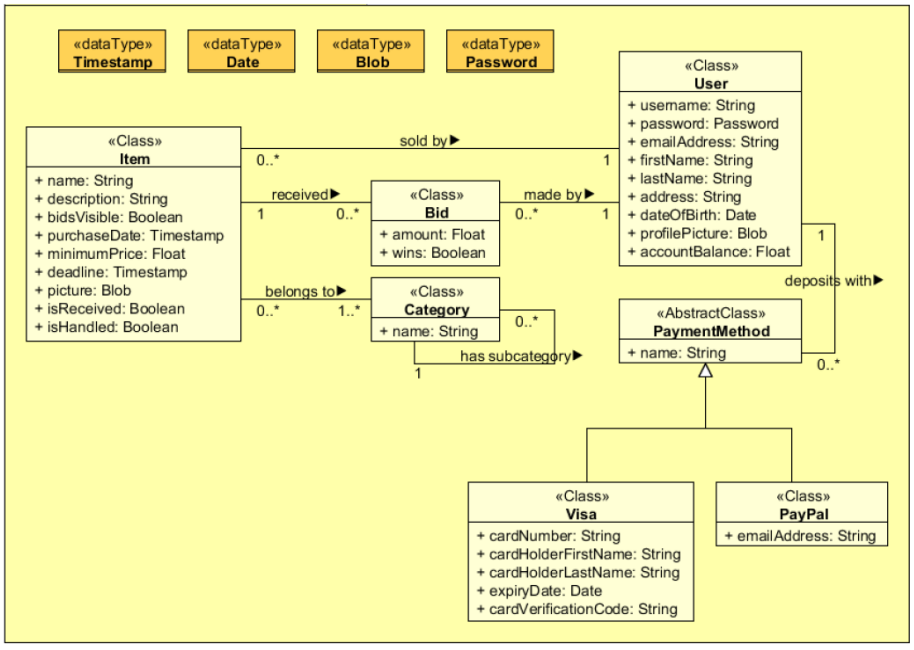


Figure 2.1: Example of a domain model, used for an item bidding website

Next to the domain model, the interaction flow model forms the core of an IFML model. Inside the interaction flow model, the content elements of the domain model can be referenced by means of *content bindings*. These can be used to indicate, for example, that a certain list user interface component will contain only elements of the type Item, which is defined in the accompanying domain model. It is, however, not mandatory for an interface component to refer to any content element.

The next subsection gives an overview of the components of an interaction flow model, how they are depicted and what their semantics are.

2.1.3 Interaction Flow Models: Syntax and Semantics

In what follows, a description of the different components that can be used to construct an interaction flow model is given, together with their IFML syntax. This components description is based on the components description in the IFML specification document (Object Management Group, 2015, p. 11-14).

View Container

A view container is a user interface element which contains other interface elements (other view containers or view components). It can group elements that can be accessed by the user at the same time, and it can also do the opposite: grouping elements which the user can only see alternatively.

Figure 2.2 depicts the four different types of view containers. Figure 2.2(a) is a basic view container that can group other user interface elements. Figure 2.2(b) is a XOR view container, denoted by "[XOR]". XOR view containers group other view containers and constrain them to be displayed alternatively. Only one view container inside a XOR view container can be shown at a time. When a XOR view container is displayed, the container's default contained view container will be displayed. Such a default view container is depicted in Figure 2.2(c). It is denoted by "[D]". Another type of view container is *landmark*, depicted in Figure 2.2(d). Landmarks can by definition be reached from any other interface element. Landmarks are denoted by "[L]".

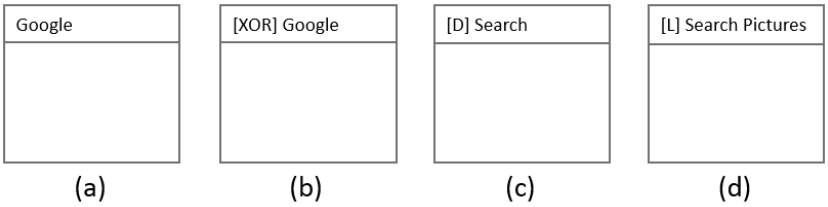
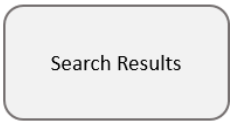


Figure 2.2: Different types of view containers

View Component

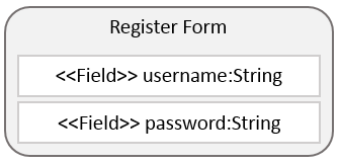
A view component is a user interface element that is contained in a view container. It can present content to the user and/or allows for interaction.

It can be, for example, a list of search result items or a contact form.



View Component Part

A view component part is an element that can only reside in a view component. It can provide more in-depth interaction details of a view component. For example, a register form that contains a password input field, where the form itself is the view component and the password field the view component part. Additionally the field could check if the input password is strong enough, by means of an action triggered by the field upon, for example, a "key-up" event.

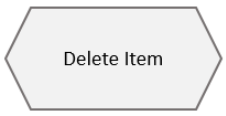


Event

An event is the result of an interaction performed by the user through the user interface or by the application itself. For example, the user pushes a search button. The application itself can also generate events, named *throwing events*. Throwing events are denoted by a black circle. For example, an asynchronous throwing event could be caught by the user interface to show the user a notification which informs him that the application has completed some action, or the application notifies the user of a deadline (the event is generated on a fixed point in time by the application).

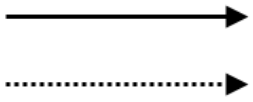
Action

An operation performed by the application behind the scenes. Actions are triggered by events. For example, delete an item from a user's profile on request of the user (the user request is the triggering event). It is also worth mentioning that an action does not have to belong to a view container.



Interaction Flow

An interaction flow carries parameters between elements in the IFML model, upon the occurrence of an event. The parameters sent in the outgoing side of the flow are used as input parameters for the element at the incoming side of the flow. There are two kinds of interaction flows: *navigation flows* and *data flows*. ***Navigation flows*** (denoted by a full arrow line) do not only carry the parameters but also navigate the user to the element at the arriving side of the flow. ***Data flows*** (denoted by a dashed arrow line), on the other hand, are only used to carry parameters around. They are often used as auxiliary flows for navigation flows when not all the necessary parameters can be found in the IFML element at the outgoing side of the navigation flow. Also worth mentioning is that navigation flows do not necessarily need to carry parameters (e.g., when the user browses to another page in a website, merely following a hyperlink). An example of a navigation flow which does carry parameters can be the following: a user submits an item using a form and arrives at a details page of the item he just inserted (carried parameters: the details of the item).



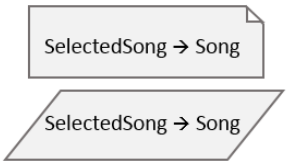
Parameter

A parameter is a variable that can be passed around by flows. It can be held by any IFML element which can have in-coming or out-going flows. For example, a view component that contains a list of songs can contain the parameter SelectedSong: when the user then selects a song from the list,

the parameter is set and can be passed onto a flow which leads to another interface element playing the selected song. Worthwhile mentioning is that parameters are not shown in the interaction flow model unless required for better understanding.

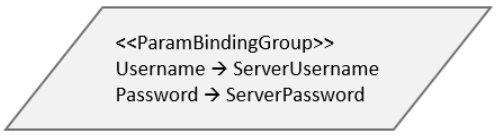
Parameter Binding

A parameter binding is the association of input and output parameters of a flow. They specify how parameters are transferred. For example, the SelectedSong parameter of a list with songs is bound to the Song parameter of a details view about the song, upon transferring it between both views by means of a navigation flow. Parameter bindings have two syntactical notations with equal semantics, and are connected to the associated flow by means of a dashed line.



Parameter Binding Group

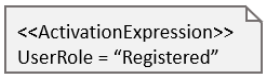
A parameter binding group groups several parameter bindings which are associated to the same flow. For example, to login to a website both parameters *username* and *password* are sent to an action (at the server-side) which checks whether the username-password combination is valid. In this example, both username and password are bound in the parameter binding group of the flow between the log-in form and the server action.



Activation Expression

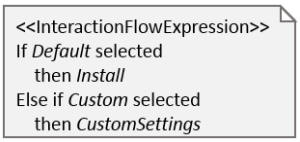
An activation expression is a boolean expression, possibly using parameters, which enables or disables an element of the interaction flow model, depending

on whether it respectively evaluates to true or false. For example, an activation expression which is used to check whether a user is logged in before transferring him to content which is only accessible for registered users.



Interaction Flow Expression

An interaction flow expression is an expression that selects the interaction flows that have to be followed. Depending on the state of the application, the interaction flow expression can select different interaction flows that have to be followed in order to present the appropriate application functionalities to the user. For example, an installation wizard which asks whether the user wants a custom install (being able to select the installation folder, etc.) or a default install (quicker than the custom install). The wizard could be based on an interaction flow expression, which guides the user to the appropriate subsequent wizard screens, based on whether the user selected the custom or default option in a combo box.



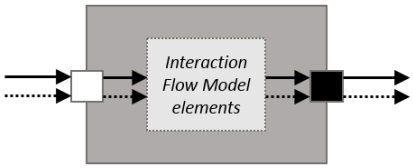
Module

A module is an element that abstracts a piece of an interaction flow model. Modules can improve the readability of an interaction flow model, because they can simplify a complex diagram. They can also improve the reusability, and therefore the maintainability, of an interaction flow model because they can be easily replaced and reused at a different place in the model. This supports an important characteristic of a platform-independent model: to be able to cope with changing requirements. Modules contain a set of other interaction flow model elements. All interaction with modules happens through their input and output ports, which can respectively receive and start interaction flows.

Ports

A port is an element belonging to a module, which allows to input (*input port*) or output (*output port*) interaction flows and parameters to and from

the module. On the inside of the module, incoming interaction flows and parameters can be used by the elements in that module. A module can have as many input and output ports (respectively denoted by white and black squares) as needed to interact with its context.



2.1.4 IFML Extension Mechanism

It is possible to make extensions to IFML, using the UML extensibility mechanisms (Object Management Group, 2015). The IFML specification (Object Management Group, 2015) provides a list of example extensions to the core components of IFML (p. 14-16). Note that these extensions cannot modify the initial IFML components. They can only make them more specific. For example, an event can be specified as a *selection event*, an event that is produced by the user by pushing a button to select some item in the user interface. In this case, the selection event is an extension to the event component. Another, broader, extension can be found in Brambilla, Mauri, and Umuhoza (2014). This article discusses an IFML extension geared towards mobile applications front-ends. It contains mobile-specific IFML components, such as *BatteryEvent* and *ConnectionEvent*, *CameraAction* or *MicrophoneAction*. This extension illustrates that IFML models do not need to stay highly platform-independent. They can be made mobile-specific, yet mobile platform independent (i.e., not dependent on whether the mobile application is an Android, Windows, or iPhone application).

Extension of the IFML components can be useful to relate the model components more easily with known user interface components —this possibly reduces the time needed to model the IFML diagram and consequentially the user interface. For example, the *List* extension, found in the IFML specification (Object Management Group, 2015, p. 15), provides the user interface developer with the information that the data inside that (List) view component should be presented in a sequential order. Note that it can still be

visually represented in different ways (e.g., as a linear list or as a group of tiles).

Furthermore, not all IFML components can be extended. The list of extendible components can be found in Object Management Group (2015, p. 10-11).

2.1.5 IFML Examples

In this section, examples of interaction flow models are presented and discussed. For the purpose of making the connection between IFML and the user interface clear, two examples will be constructed in different ways. The first example will start by creating the interaction flow model and then suggesting an appropriate user interface for the modelled application, while the second one will start with a given user interface and produce a corresponding interaction flow model. The latter is also included to suggest that an existing application might be reverse-modelled in IFML, if, for example, IFML is chosen to be included in the development cycle of the application. Note that Rodriguez-Echeverria et al. (2014) digs deeper into this reverse-modelling problem and presents a specific process to generate IFML models from legacy applications, in order to modernise their front-end. Since it is out of the scope of the simple reverse-modelling example presented in this subsection, no explicit process will be used here. The IFML model will be created by hand, from scratch.

Video Player Example

Suppose one wants to model an application to playback videos, according to the following set of requirements:

1. A *list of videos*, available for playback, is presented to the user;
2. A *video display* is shown alternatively with the list of videos;
3. The user can *select a video from the list* to start the playback;
4. While the video plays, the user can *control its time line* (i.e., allowing to pause, replay, fast forward and rewind the video);
5. The user must be able to *quickly switch to the previous or next video* in the list, without explicitly selecting it from the list.

Figure 2.3 depicts an interaction flow model of a possible video player application, modelled according to the previously stated requirements. Note

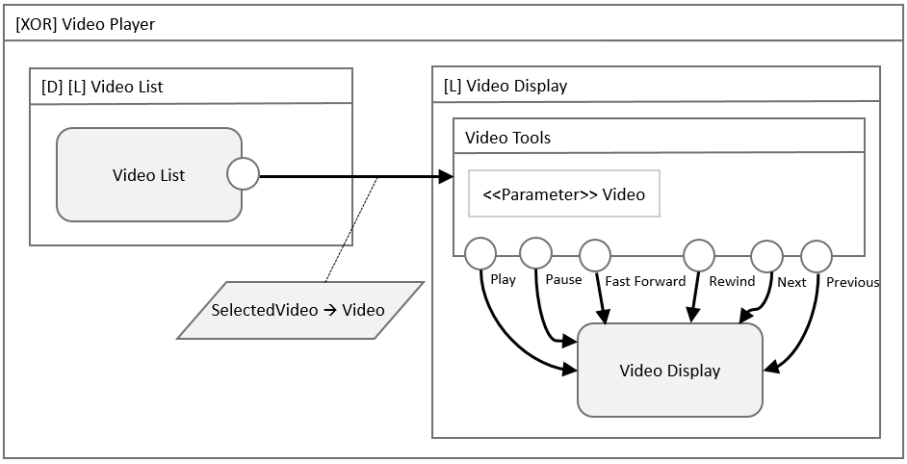


Figure 2.3: The Video Player interaction flow model

that for this example, it can be assumed that the accompanying IFML domain model merely consists of only one class, which is *Video*.

From requirements 1 and 2, one can deduce that the application should display two views alternatively, the video list view and the video display view. In IFML, the XOR view container is suited to represent this, see the *Video Player* container in Figure 2.3. Its two contained view containers represent both the video list and video display views, *Video List* and *Video Display* respectively. *Video List* is the default container, which means concretely that, at start up, the application will only show the video list.

Inside *Video List* there is a view component with the same name representing the actual list that will be visible in the user interface. At first this seems redundant. However, it is needed to include the video list in the surrounding *Video List* view container to be able to show the video list and the video display alternatively, using the XOR view container.

To fulfil requirement 3, an event with an outgoing navigation flow and parameter binding is attached to the video list. This event is produced when the user selects a video of the list. The event navigates via the flow to the video display, where the video then starts to playback. Once the user arrives there, he can control the video's time line and switch to the previous or next video with the provided *Video Tools* (cfr. requirement 4 and 5). Several events attached to the tools component provide the means for this interaction. One can discuss the need of the *Video Tools* view container, and wonder why the events are not attached directly to the *Video Display* container. This choice was made to make a distinction between the containing view element and the available tools related to the displaying of the video.

However, the option without the separate tools container would be correct too.

Figure 2.4 presents a user interface designed according to the interaction flow model of Figure 2.3. Figure 2.4(a) shows the list view and Figure 2.4(b) shows the display view. Both views can be accessed using the tabs at the bottom of the player. These tabs are created from the landmarks in the interaction flow model. As can be seen in the list view, a linear scrollable list with the video titles was chosen. In the display view, the modelled events can be triggered by three buttons: the play button in the middle, which transforms into a pause button when pushed, and two smaller buttons next to the play button. The latter ones allow to navigate to the previous and next videos, and to fast forward or rewind when held down for a longer time[3]. Using just three buttons which cover six events illustrates that not all components in the interaction flow model are necessarily separate components in the user interface. Those design choices are entirely made by the user interface designer and not by the interaction modeller.

Android Website Example

This example illustrates how the Android website[4] can be modelled in IFML. As previously mentioned, it will be shown here that IFML can be used to model the interaction design of an already existing application.

Figure 2.5 depicts the Android website home page. When the user clicks the magnifying glass in the upper right corner, the navigation bar on top of the page transforms into a search field in which the user can write a search text, depicted in Figure 2.6.

In Figure 2.7, an IFML model, describing a part of the Android homepage functionality, is shown. Note that it is assumed, for this example, that IFML is extended with a *Field* view component part which is used in forms, and a *Modeless* view container which is displayed only when navigated to by a navigation flow.

The most interesting part of this IFML model is the search interaction. When the magnifying glass is clicked, the *Open Search* event is raised. It activates a search form with one field for the query string. When the user pushes enter on his keyboard, the *Search* event will try to invoke the *Search Matching Items* action. If the search query string is not empty, the action will be executed. If it is empty, nothing will happen. This gate keeping is

[3]Note that this button design is not the best design practice, yet it is commonly used in video players, such as Windows Media Player.

[4]http://www.android.com/, retrieved on 26 Oct. 2014

(a)

(b)

Figure 2.4: A possible Video Player user interface. Video data was obtained from IMDb's Top 250 (`http://www.imdb.com/chart/top?ref_=nv_ch_250_4`, retrieved on 26 Oct. 2014)

performed by the activation expression. Once search results are obtained by the remote server, a new page with the search results will be loaded.

Note that, according to the IFML model, the top menu bar items, as displayed in Figure 2.5, are not disappearing when the search magnifying glass is clicked. According to us, this case is very difficult to express using the IFML core constructs, since the menu bar items should be expressed as landmarks that disappear in certain occasions. This would contradict the definition of landmarks. One can argue about the user interface design of the

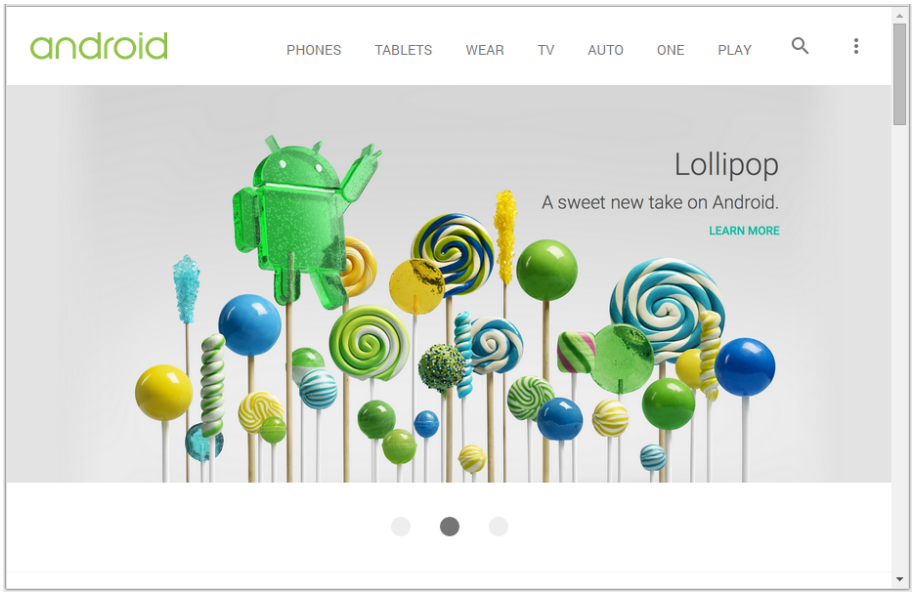


Figure 2.5: Android website: homepage user interface

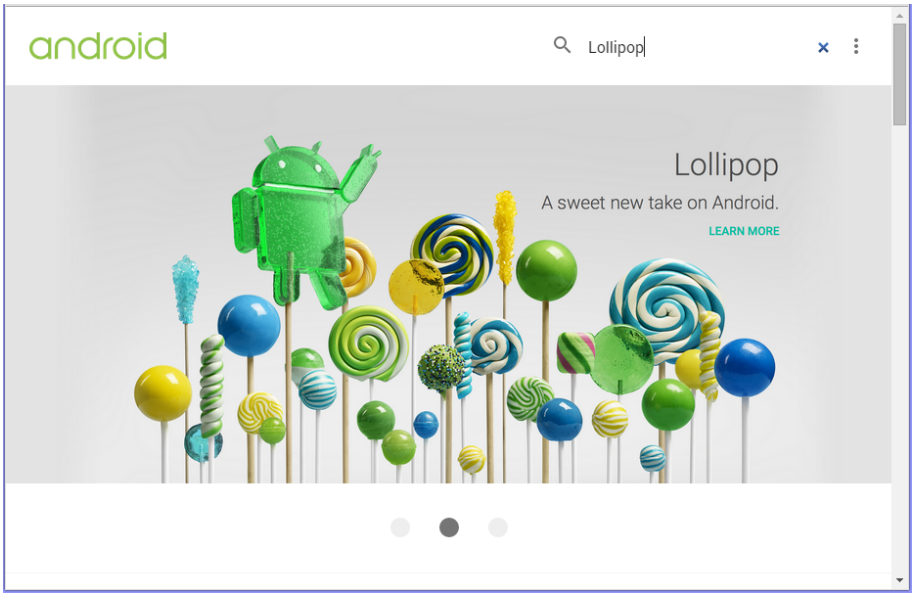


Figure 2.6: Android website: search bar visible

Android website. It seems rather bad practice to hide the menu bar behind a search field.

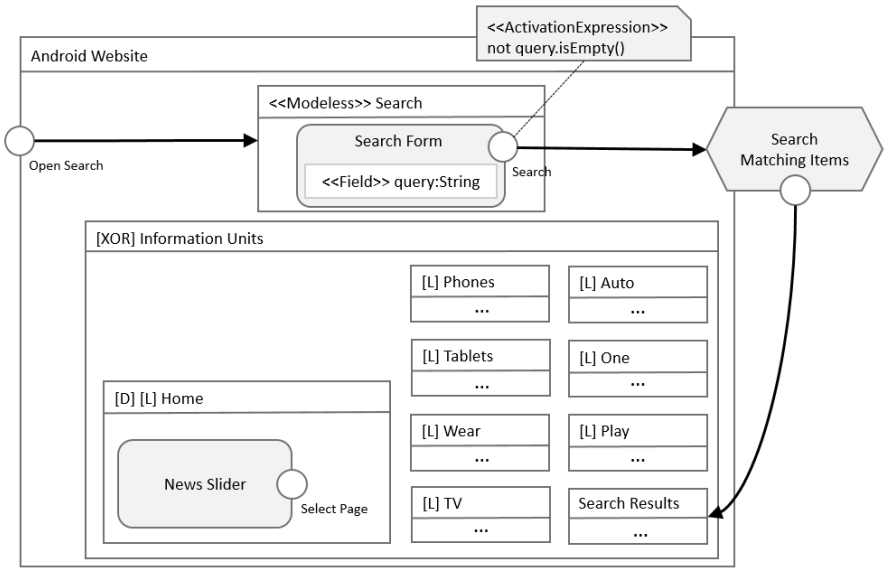


Figure 2.7: The Android homepage interaction flow model

2.1.6 A Few Notes on the IFML Semantics

As the previous Video Player example shows, it appears as if an interaction flow model can be syntactically different, yet semantically equal (in the Video Player example: grouping of the tools in an extra view container or not). This is our interpretation of the IFML specification (Object Management Group, 2015). The specification document does not specifically mention that a semantic difference is present when events are modelled attached to a view container or when they are modelled attached to a view container inside another view container. This syntactical difference is shown in Figure 2.8. Note that this holds for the model parts shown in this specific figure. It will not necessarily hold when other elements are added to the containers.

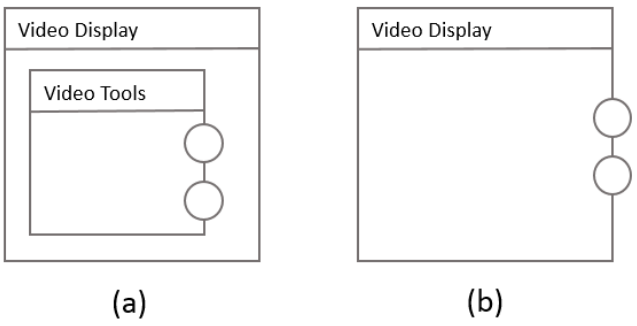


Figure 2.8: Two syntactically different IFML model parts

What is also worth mentioning is that the interaction flow model core components, in certain cases, can impose constraints on the user interface. This is, for example, the case when using XOR view containers in a specific way. When such a container is used to express that a user is not allowed to see two elements inside the container at the same time, it does not impose any decision that should have been made by the user interface designer, since it is an application requirement. However, when the XOR container is used for, for example, reducing the information load on the user's screen, it is in fact specifying a user interface aspect, which is normally taken care of by the user interface designer.

2.2 Web Semantics Design Method (WSDM)

The *Web Semantics Design Method* (WSDM) is a method to develop websites or web applications. Originally, it was named *Web Site Design Method*, and could be used to develop informational websites only (De Troyer & Leune, 1998). However, as the World Wide Web evolved into the Semantic Web (Berners-Lee, Hendler, & Lassila, 2001), WSDM got adapted to deal with web applications and to be able to incorporate semantic annotations in the developed web systems (Plessers, Casteleyn, & De Troyer, 2005).

This section gives an overview of WSDM. After an introduction to WSDM, it starts by explaining the method's structure at a high level and then goes into greater detail, discussing all of its steps one by one. If more information would be required, one can read the complete description of the Web Semantics Design Method in De Troyer et al. (2008).

2.2.1 Introduction to WSDM

WSDM distinguishes itself from other website design methods by being *audience-driven*. This means that WSDM incorporates the fact that websites might have different groups of visitors, which all have their own set of requirements (De Troyer, 2001). These groups are called *audiences*. By targeting those audiences with a website fit for their needs, it is more likely that they will have a satisfying experience using the website and that they will return to it in the future (De Troyer, 2001). When, for example, a bank wants to create a website to provide information about bank accounts and loans to private persons, as well as information about investments and real estate to businesses, two different audiences could be determined: private users and professional users. If then the audience-driven approach is precisely followed, the audience of private persons will be able to, for example, quickly open a

new savings account or request an appointment with the nearest banking office to discuss a loan. On the other hand, professional users would have direct access to information about interested investors and real estate management possibilities.

Who is acquainted with *user-centred* approaches, might wonder how those differ from an audience-driven approach. As opposed to a user-centred approach, an audience-driven approach does not involve end-users in the website development process. Involving users is not always possible because they are not known upfront when developing websites for the Web (De Troyer, 2001).

WSDM is a method which proceeds in different phases, as shown in Figure 2.9. In the first four phases the website is modelled, based on the requirements of the different audiences. The final outcome of these phases is a set of models and descriptions which is used for the implementation of the website (this corresponds to the fifth phase, Implementation). Ultimately, the Implementation phase could be supported by an automated tool which turns the models of the previous phases into an implementation of the web system. Such a code generation tool does however not exist at the time of writing. As WSDM is an iterative process, it is possible to return to any of the first four phases, adapt them, and percolate these changes further down the process to at last implement the adjusted website.

2.2.2 Phases

This subsection will explore the different phases of WSDM, and is therefore largely based on the complete WSDM description (De Troyer et al., 2008). Figure 2.9 shows an overview of those phases (and their sub-phases) and how they are sequentially ordered. In this figure, phases are denoted by rectangles with a white background and sub-phases consist of rounded rectangles with a gray background. As can be seen, WSDM is a 5-phase process. The phases are, consecutively: *Mission Statement Specification*, *Audience Modelling*, *Conceptual Design*, *Implementation Design* and *Implementation*. Each phase in the process takes the output of the previous phase(s) as its input.

Phase 1: Mission Statement Specification

Output: a textual mission statement

In this phase, the *mission statement* of the web system is specified by means of natural language. It resembles a mission statement that is often specified for businesses. In the case of businesses, the mission statement is often

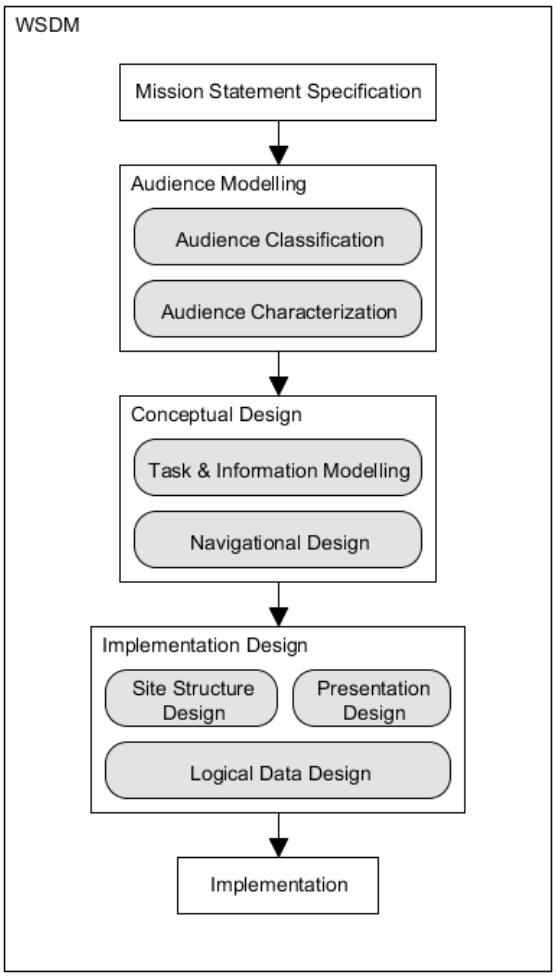


Figure 2.9: Structure of WSDM (Based on De Troyer et al. (2008))

used as a guideline for the employees to keep in the back of their minds when taking decisions in the company. This is much in line of what the WSDM mission statement wants to achieve, namely: serving as a basis for the process to guide the design decisions taken in the process. It can also be used to evaluate the website: checking if the derived implementation is in accordance with the original purpose of the website (De Troyer et al., 2008). To guide the writing of the mission statement, three aspects are specified which need to be included explicitly in the mission statement: the *purpose*, the *subject* and the *target users* of the web system (De Troyer et al., 2008).

Suppose a development team needs to build an online website to share

videos, such as, YouTube[5]. An example mission statement could be the following:

> *To provide a large user-driven online platform for sharing videos and commenting on videos, delivering videos at a high speed and a high quality. Any online user can watch as much videos as he/she likes and subscribed users can post an unlimited amount of videos for others to watch.*

In this mission statement the purpose of the web application is: providing a large user-driven online platform for sharing videos and commenting on videos, while delivering them at a high speed and a high quality. The subject is clearly videos and their associated comments given by the users, and the identified target users are users which do not subscribe and merely watch videos ("video watchers") and users which do subscribe and thus can post videos for others to watch ("video posters").

Note that the mission statement is not the entire specification of the web system and is therefore incomplete. In the next phase, requirements will be specified to complete the web system specification.

Phase 2: Audience Modelling

Input: the textual mission statement

Output: audience hierarchy, audience requirements, audience characteristics (collectively referred to as the *audience model*)

This phase is concerned with creating a set of different audiences for the web system under development, based on the target users specified in the mission statement. During the next phases of WSDM, the resulting audiences will be taken into account separately, so that the web system serves each of them according to their needs. As can be seen in Figure 2.9, this phase consists of two sub-phases, which are considered sequentially: *Audience Classification* and *Audience Characterization*.

Audience Classification

In this sub-phase, *audience classes* are created based on the identified target users. An audience class is a group of users which have the same informational and functional requirements. When a group of users would have a subset of the requirements of another group of users, this latter group is not

[5]https://www.youtube.com

regarded as separate audience class, but rather as an *audience subclass*. Audience subclasses thus have extra requirements in addition to those specified for their superclass. Important to note is that a subclass involves part of the population of its superclass.

WSDM provides two methods that can be used to find the different audience classes of the web system under development, both specified in De Troyer et al. (2008). One method starts by listing the activities of the organisation related to the web system. Then it deduces the informational and functional requirements of the people involved in those activities and separates them in different audience classes based on the differences in the deduced requirements. The other method builds audience classes based on a matrix of all requirements of all target users. It then compares the rows of the matrix, looking for equivalent ones. The users associated to these rows can belong to the same audience class. A more detailed description of the two methods can be found in De Troyer et al. (2008) and Casteleyn and De Troyer (2002) (for the second method). Both methods require to specify the informational and functional requirements of all the audience classes along the way to find those classes. When all audience classes are determined, other requirements (usability or navigational requirements) should be added to the different audience classes. The total set of requirements together with the mission statement now compose the overall system specification.

Figure 2.10 shows an example of a derived audience class hierarchy for the video sharing website mentioned in the example in Phase 1. This hierarchy is written in the notation specified by De Troyer et al. (2008). Each human stick figure represents an audience class and the arrows connect subclasses to their superclasses. As one may notice, the top-level audience class is called "Visitor". This is standard in WSDM and cannot be adjusted. The Visitor class represents requirements which all target users have in common (De Troyer et al., 2008). In the example hierarchy, "Video Watcher" and "Video Poster" (cfr. the target users in Phase 1) are two separate audience classes. One can wonder why video watchers would be a separate class, since, keeping the example of YouTube in the back of the mind, everything a video watcher can do, a video poster can do too. The difference lies in the fact that the users in both audience classes visit the website with a different intent. Video watchers are concerned about whether the video can be expanded to full screen, or whether it is available in high definition. This is not so much the concern of a video poster. One can also look at it this way: when the video poster is done with posting videos and decides to go watch some videos instead, he changes audience class and becomes a video watcher, with different requirements.

In the same figure, a possible audience subclass "Company" is shown. This class could represent video posters with an extra requirement: advertising for their own videos inside other videos on the website.

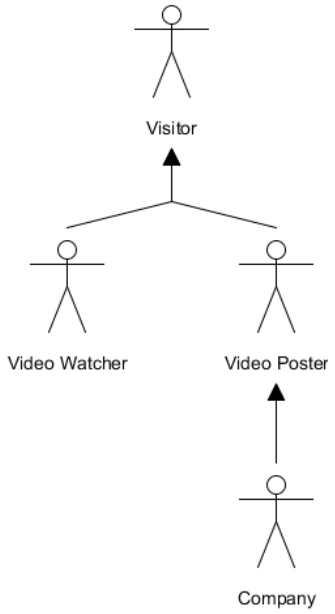


Figure 2.10: Audience class hierarchy for the video sharing website example

Audience Characterization

This sub-phase consists of specifying relevant characteristics for the identified audience classes. These can be used later on in the process (e.g., in the Implementation Design phase) to guide the visual design of the web system (De Troyer et al., 2008). Characteristics can include, but are not limited to: frequency of use, level of experience with websites, level of task knowledge, age, type of user (direct or indirect), mandatory or discretionary use of the system. For example, the average video poster will have more experience with websites than the average video watcher (since they will need to set up an account on the website, etc.).

Phase 3: Conceptual Design

Input: audience classes and requirements

Output: CTT task models with associated ORM object chunks, navigational model

Now the requirements of the web system to be build are elaborated upon in the previous phase, it is time to specify them more formally, by means of conceptual models. The *Conceptual Design* phase focuses, as its name suggests, on these conceptual models. In this phase, it is important that no direct connection is made with any implementation-specific concepts. The conceptual models are thus free from any reference to existing technologies or systems. This separation of concerns allows, for example, to change the underlying technologies of the existing system more easily in a next iteration of WSDM, since the conceptual models of the application are already in place, prescribing all functionality of the web system, and do not need to be adjusted.

The Conceptual Design phase exists of two steps: *Task & Information Modelling* and *Navigational Design*. The first step has as output the *task models* and *information models*, specified in the CTT and OWL (or ORM) notations (these notations will be elaborated upon further in this phase). The task models specify the tasks which users have to be able to perform, while information models describe the information (or "content") available to the users. The second step has as output the *navigational models* specified in a language defined by WSDM itself. This model describes how the users can navigate through the web system and where along that navigation path they can perform which tasks.

Task & Information Modelling

This sub-phase starts by creating task models for all the tasks that the web system needs to support (*Task Modelling*). Phase 2, Audience Modelling, provided all the informational and functional requirements of the system, per audience class. Now, for each requirement, tasks will be constructed that satisfy the requirement. This results in a set of task models for each audience class. Since it is essential that these tasks are constructed formally and unambiguously, a specific task modelling notation, *ConcurTaskTrees* (CTT) (Paterno et al., 1997), will be used for this. CTTs have the ability to decompose each high-level task into elementary tasks, specifying exactly how the user should be able to perform a task with the system. It can express, for example, that the user needs to be able to carry out two tasks in parallel or sequentially, or whether a task can be repeated several times. CTTs do this by using *temporal operators*: operators which express temporal relationships (Paterno et al., 1997). These temporal operators will be explained further on,

in a CTT example. In WSDM an adapted version of CTT is used. Among other differences, which can be found in De Troyer et al. (2008), the WSDM CTTs do not provide as much details as original CTTs do. These details will namely be provided by the information models, which are subject further on in this sub-phase.

In the video sharing website example, the following requirement can hold for the Video Watcher audience class: "video watchers should be able to comment on videos while watching them". Figure 2.11 depicts what this requirement would look like when it is translated formally in a CTT tree. Each node in the tree represents a task. The top node is an abstract task ("Interact with Video") which exists of two sub-tasks ("Watch Video" and "Comment on Video"). To complete the abstract task, at least the "Watch Video" task needs to be completed, as the "Comment on Video" task is optional (denoted by the square brackets "[ ]" around its name). These two sub-tasks of "Interact with Video" can be executed concurrently, denoted by the temporal operator "|||" in between them. This means the user can comment on the video meanwhile he is watching it. The left sub-task is also an abstract task since it contains two other tasks: "Play Video" and "Pause Video". These are *interaction tasks*, tasks that represent interaction of the user with the system[6]. The "Comment on Video" task is also an interaction task. "Play Video" and "Pause Video" are tasks that have to be performed alternatively. This is represented by the "[ ]" symbol (the *choice* operator) in between the two tasks. The star ("*") behind the "Watch Video" task name makes sure that the user can repeatedly play or pause the video. The star thus represents iteration.

The second part of this sub-phase, *Information Modelling*, is about constructing pieces of information which the user needs for each of the tasks inside the task models. Every elementary task is assigned the needed information in the form of an *object chunk*. These object chunks are, often small, information descriptions in *Web Ontology Language* (OWL) syntax[7]. Using this Semantic Web technology enables easier generation of semantic annotations in the last WSDM Phase (Implementation) (De Troyer et al., 2008). However, because OWL does not yet have a graphical notation which suits WSDM, the *Object-Role Modelling* (ORM) notation (Halpin & Morgan, 2008) is used. ORM was chosen because it can be mapped easily to OWL (De Troyer et al., 2008).

In Figure 2.12, an example object chunk is presented for the "Play Video"

[6]In WSDM, *application tasks* (tasks performed by the application) are another type of tasks which can be specified in CTTs. These will however not be discussed here.

[7]http://www.w3.org/TR/owl-features/

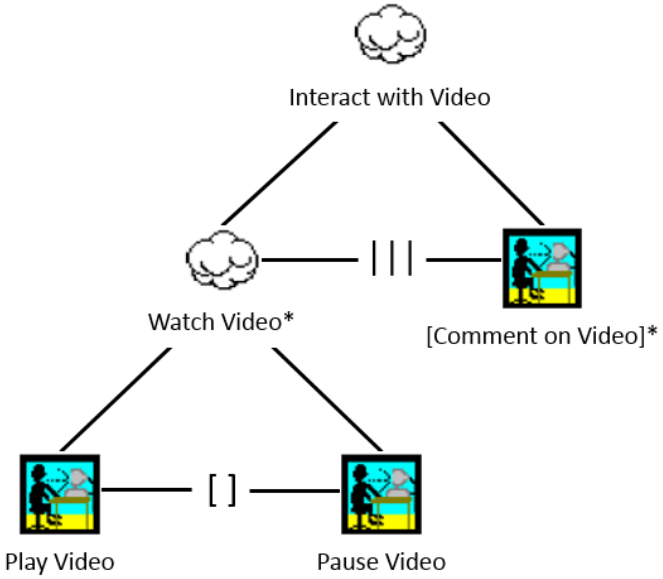


Figure 2.11: ConcurTaskTree, representing a requirement of the video sharing website example (Icons adopted from Paterno et al. (1997))

elementary task (specified in Figure 2.11). To allow communication between tasks, input and output parameters can be specified in the right lower corner of the object chunk. The parameters represent instances of object types in the ORM schema[8]. The same instances are also depicted on the ORM schema inside the object chunk. This makes it possible to model functional requirements on them. To achieve this, WSDM extends ORM with an expression language inside the ORM schema. In Figure 2.12, these expressions are depicted in blue. The **v* input instance is a Video instance which is modified in this task. When the video is playing the current duration increases. Therefore, the *Duration* instance (**d*) is updated during this task, denoted by the expression $*d = playtime$[9], where *playtime* is a system function that captures the time played. After task completion, the modified **v* instance is outputted again and carried on to the next task which is executed. Note that the exact same object chunk as presented in Figure 2.12 can be used for the "Pause Video" task in Figure 2.11. The user needs the same information

[8]Object types are represented by rounded rectangles with a full border line. They can be seen as a class, which is related to other classes (other object types) or values. For more information about ORM and the ORM2 notation, one can read Halpin and Morgan (2008).

[9]Note the use of the assignment operator here. WSDM provides many more operators to express other functionality. De Troyer, Casteleyn, and Plessers (2005) describes all of them.

for both tasks.

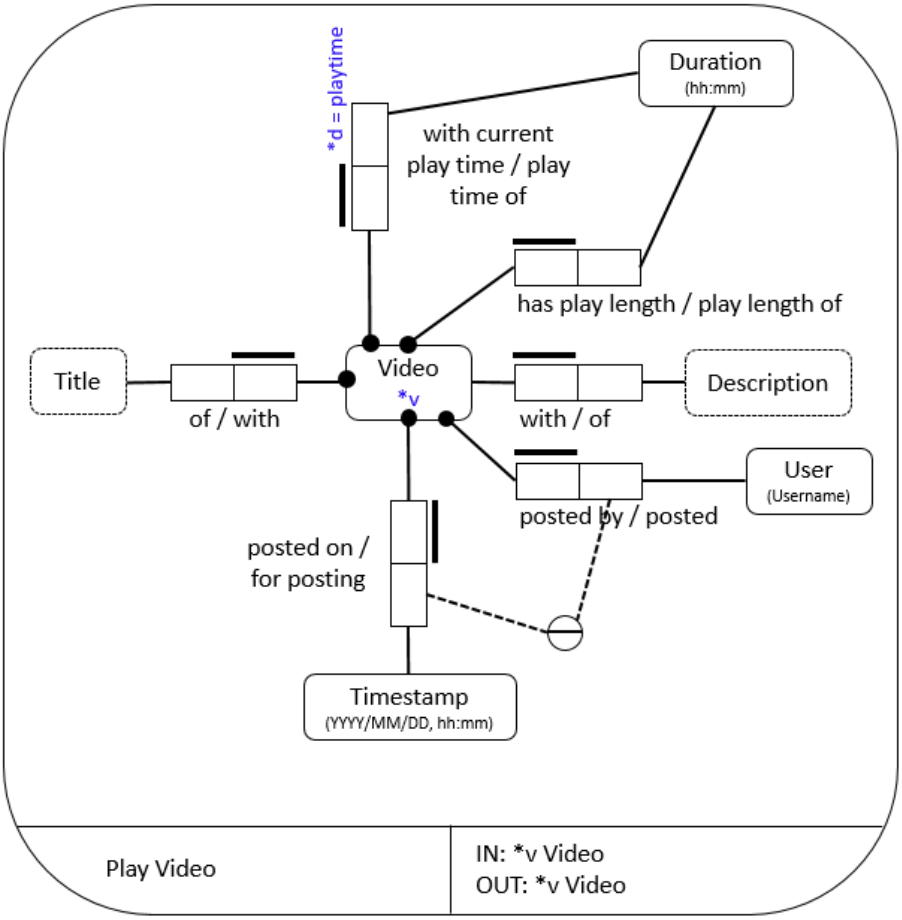


Figure 2.12: Object chunk example (ORM2 notation)

Navigational Design

In this second sub-phase of the Conceptual Design phase, the *navigational model* is created. This model is a composition of different *audience tracks* which represent how users of a specific audience class can navigate around on the website and when they can perform which tasks. To construct this model, the task models from the previous sub-phase are converted in *task navigational models*. For each CTT model, a task navigational model is constructed. To construct such a model, first the elementary tasks in the CTT model are converted into *components*, and then the temporal operators are converted into *process logic links* between the components. Object chunks are then connected to each component to make the links between tasks and

the information needed for these tasks visible.

An example of a navigational model is given in Figure 2.13. At the top, the highest level of the model is shown: the navigational model. It exists of the homepage of the website which leads to the navigational tracks of the watcher and the poster audience classes. Note that posters first need to pass a login step. The second level presents a zoom-in on the video watcher audience track. The video watcher first searches for a video and then interacts with it. Finally, at Level 1, the task navigational track is presented for the "Interact with Video" task. This is the most detailed level of the navigational model. It shows the flow of the task, taking also the ORM object chunk parameters in account, which flow to and from components. Also the object chunk itself is linked to its associated component. Note that not all parameters and object chunks were added in the presented figure to prevent overloading this figure.

Navigational aid links can be defined to enhance the web system's usability (De Troyer et al., 2008). They are added to the navigational model in the end. Examples of these links are a homepage link to navigate from anywhere back to the website's homepage, or landmarks which are pages which can be reached from anywhere else in the website (see the "H" and "L" icons on the navigational model in Figure 2.13). Also, in addition to the navigational model, a *semantic link model* can be constructed. In this model, semantic links between different object chunks can be defined. This results in extra navigational links in the generated web system (De Troyer et al., 2008).

Phase 4: Implementation Design

Input: navigational model, user characteristics, usability requirements

Output: site structure model, templates, page models, styles, data source mapping (, logical data schema)

To be able to implement the web system at the end of the WSDM process, the conceptual models of the previous phase must be complemented by implementation-specific models and designs. These provide instructions for a concrete implementation of the web system, with the necessary looks and flow. The *Implementation Design* phase consists of three sub-phases which construct the implementation-specific models and designs: *Site Structure Design*, *Presentation Design*, and *Logical Data Design*. These phases respectively tackle: structuring the web system using pages, providing an appropriate look and feel, and mapping the presented data to an existing data source.

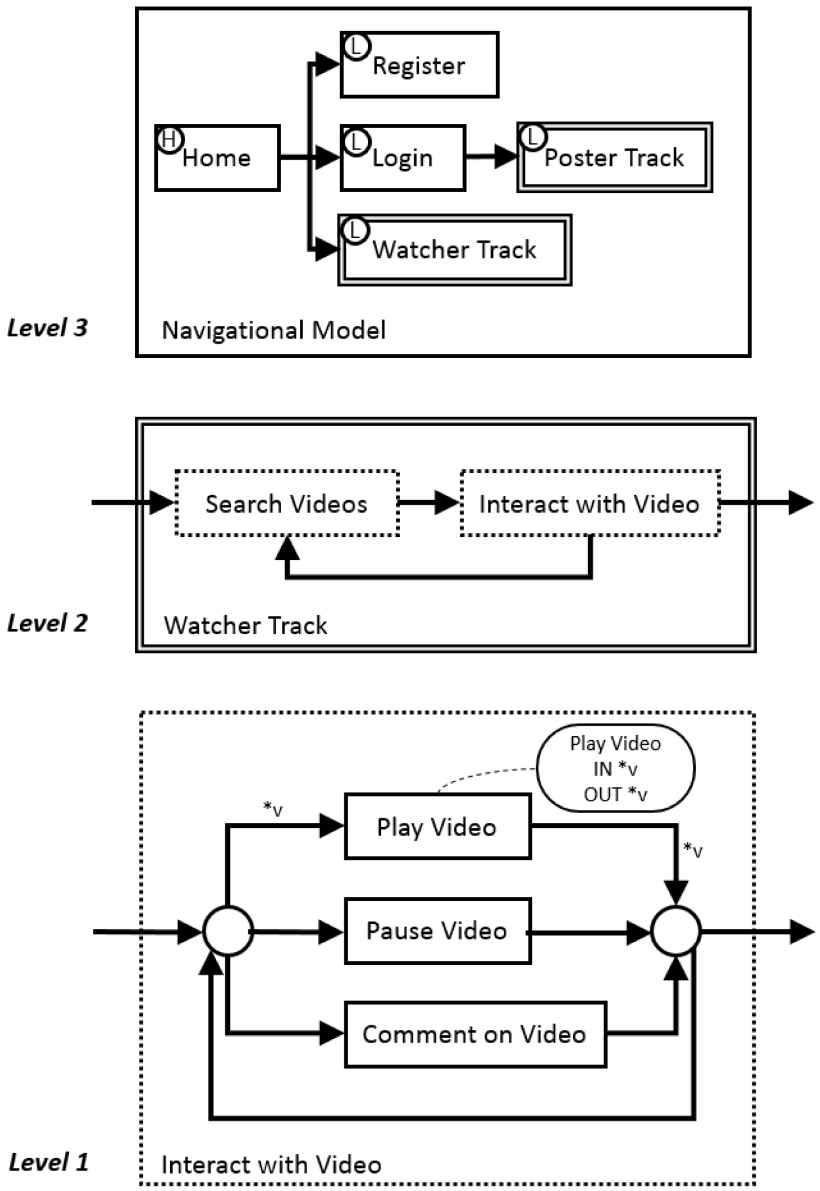


Figure 2.13: Navigational model example showing the three different levels of detail

Site Structure Design

In this first sub-phase of the Implementation Design phase, the web system is structured into pages. A *page* contains elements of the web system which can be seen at once by the user. To obtain such a page structure, the navigational

model, constructed in the previous phase, is taken as input and enhanced with a page layer. This enhanced model is called the *site structure model*. The site structure model serves the purpose of, for example, grouping several components in the web system —and thus reducing the number of links the user has to click to get to other components—, or spreading a user task over several pages to decrease the amount of information on a single page.

Figure 2.14 shows a site structure model based on the navigational model example of the previous phase (Figure 2.13). Level 3 of the navigational model is retaken and overlaid with a page structure[10]. The home component and the search component inside the watcher track are grouped onto one page. This means that the user will be able to search for videos directly on the home page. If he found one he would like to watch, he selects it and gets transferred to another page on which he can interact with the selected video. The other components in the site structure model, which are not grouped by a page symbol, reside alone on a single page (e.g., a registration page).

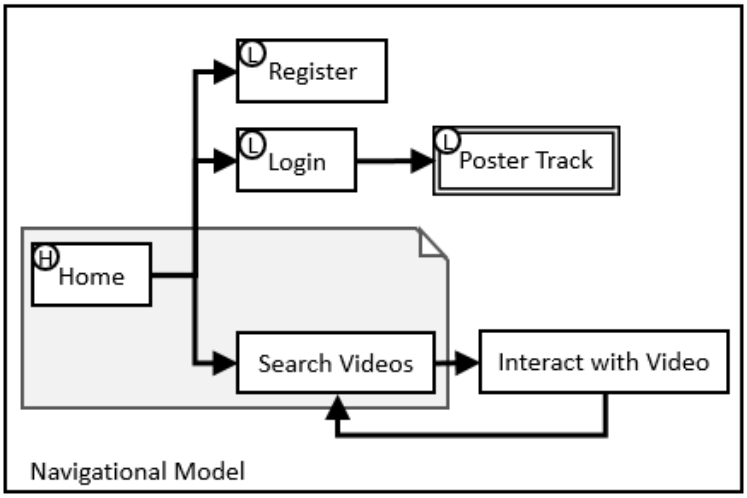


Figure 2.14: Site structure model, based on Level 3 of the navigational model in Figure 2.13

If a web system needs to be accessible on different devices with different screen sizes, typically one site structure model is created for each device type (De Troyer et al., 2008). For example, the smart phone version of a web system will often contain less components per page than the desktop version.

[10]Note that in Figure 2.14 Level 2 of the navigational model is exposed inside the presented Level 3.

Presentation Design

This sub-phase deals with the presentation of the web system to its users. Its goal is to obtain a set of page designs for each of the pages following from the Site Structure Design. These designs must take into account any audience characteristics and usability requirements which were specified in the Audience Modelling phase. For example, if colour blind persons are part of the target audience, page designs should avoid using the colors red and green.

The Presentation Design phase outputs two design models in specific: *templates* and *page models*. Templates capture the look and feel of the web system which is consistent over several pages. They leave blank certain regions on the page of which the content varies from page to page. Those regions are called *editable regions*. The page models on their turn are created for each page that is defined in the site structure model. They take the template associated with a page and specify which data and user interface elements should be presented in the template's editable regions. To specify data in an editable region, the page model must refer to elements in the object chunks associated with the components that reside on the page. To specify the user interface elements, several predefined presentation modelling concepts can be used (e.g., menus, figures, radiobuttons, on-click events; for more information on those, see De Troyer et al. (2008)). Note that templates can use the exact same concepts and object chunk references as page models.

Those presentation modelling concepts must however be visually structured in the templates and page models. This is taken care of by adding them to a *grid*. The grid defines the position of the elements inside it, their width and their height. In Figure 2.15, an example template for the "Home + Search Videos" page (cfr. Figure 2.14) of the video website is shown. The black lines that overlay the template design form the grid. The dashed rectangle depicts where the editable region of the page is. This editable region is filled in by the page model in Figure 2.16. Once videos are looked up by using the search bar, the results pop up in a list containing a representing image for the video, its title, duration and description. An on-click event could then be set on the video title to send the user to the page where he can watch the video.

The coherent looks of the pages are defined by *styles*. This allows reusing the looks across different templates and page models. In WSDM, styles are defined in *Cascading Style Sheets*[11] (CSS), which is a technology most commonly used on the web.

[11] http://www.w3.org/Style/CSS/

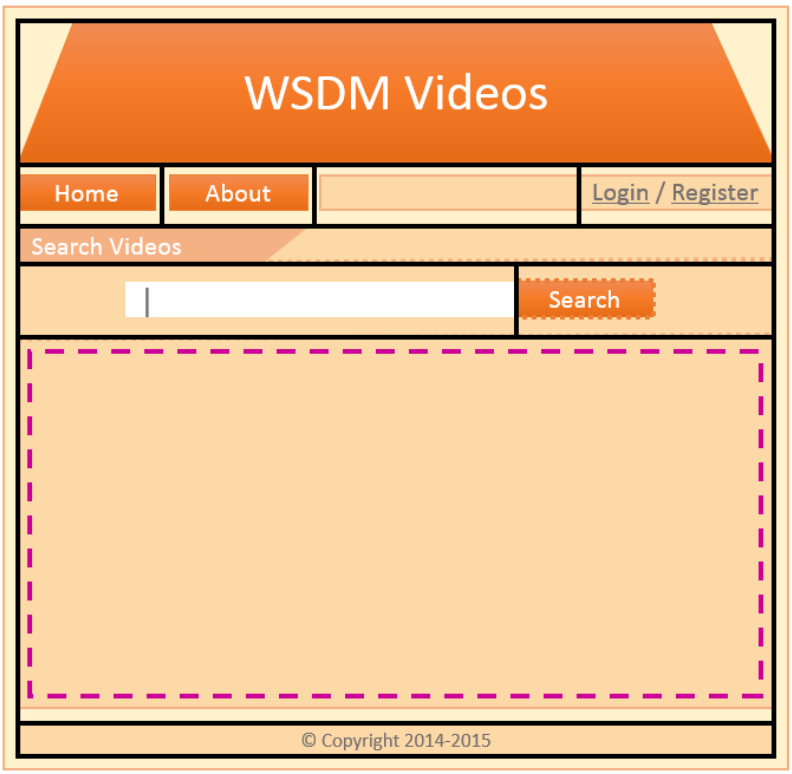


Figure 2.15: Template for the home page of the video website, including its grid

Logical Data Design

This last sub-phase of the Implementation Design phase is used to generate a mapping between the object chunks defined in the Conceptual Design phase and the logical data schema that is defined by the data store on which the web system's data will reside. If such a data store is not yet present, a logical data schema can be generated from the *reference ontology* —in order to build the data store. The reference ontology comes forth of the created object chunks. It gathers all the different concepts in the object chunks and links them. For example, when a Video object type is used in two different ORM object chunks, the reference ontology will collect the Video concept and ensures that both Video object types are specifying the same type of application object. When the data store to be used already exists, it merely needs to be mapped to the reference ontology. Consequently, this sub-phase outputs a data source mapping and optionally a logical data schema.

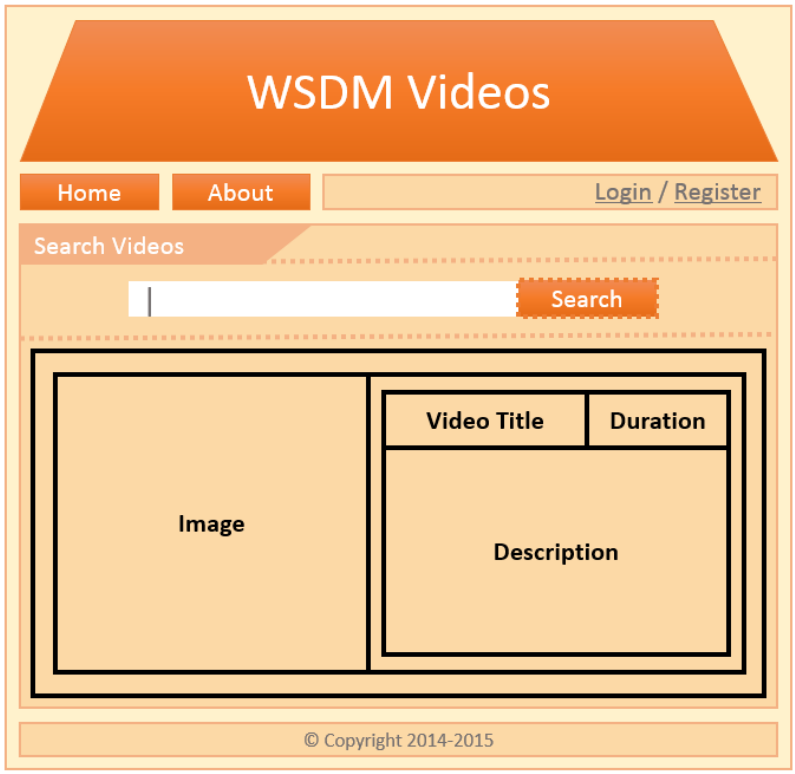


Figure 2.16: Page model for the homepage of the video website

Phase 5: Implementation

Input: object chunks, navigational design, site structure model, templates, page models, styles, data source mapping (, logical data schema)

Output: a (semantically annotated) web system

This last phase handles the implementation of the modelled web system. As mentioned before, this can ultimately be done automatically, without manual programming. Although, no tool exists for this so far. Both cases —manual and automated implementation— take as input the previously constructed design models. In the Implementation phase, semantic annotations for the web system's content can be generated too. More about this in the following subsection, Semantic Annotations.

2.2.3 Semantic Annotations

As mentioned before, WSDM was revised to comply to requirements imposed by the Semantic Web. It uses OWL (or ORM, which can be easily mapped

to OWL) for constructing object chunks in the Conceptual Design phase (i.e., the reference ontology), but also for describing the output of WSDM itself. This is done by using a predefined OWL ontology, called the "WSDM ontology"[12]. It supports the different WSDM concepts, such as audience classes, requirements, pages and grids. The outputs of the WSDM phases are described according to this ontology and are then input (including the reference ontology) in the implementation generation pipeline which uses them to generate semantic annotations at the end of this pipeline. Plessers et al. (2005) describes such a generation pipeline. WSDM thus uses semantics inside its process to more effectively generate semantic annotations of web page content and structure at the end of its process.

[12]http://wise.vub.ac.be/ontologies/WSDMOntology.owl

3 Related Work

This chapter gives an overview of related work. First, *WebRatio*, a commercial tool that implements an adapted form of IFML, is discussed. After that, other web design methods are mentioned and compared with WSDM.

3.1 WebRatio

WebRatio[1] (Acerbis et al., 2004; Acerbis, Bongio, Brambilla, & Butti, 2007) is a commercial web development tool, which focuses on fast development through model-driven development and code generation. WebRatio was founded in 2001 and was initially based on the *Web Modelling Language*[2] (WebML) (Ceri, Fraternali, & Bongio, 2000). Later, IFML, discussed in Section 2.1, was adopted in WebRatio to replace WebML. WebRatio is currently the only web development framework implementing IFML.

In the following subsections, WebRatio will be discussed in greater detail. It will be shown how WebRatio is used to build web applications from IFML models, and how exactly IFML is integrated in WebRatio. Note that there will be mainly focused on the integration and support of IFML in WebRatio, as this is what relates WebRatio to the subject of this thesis.

[1] http://www.webratio.com/
[2] http://www.webml.org/

3.1.1 Development with WebRatio

WebRatio supports the complete process of modelling of a web application. One typically starts by creating the data model for the application, called the "domain model" in WebRatio (cfr. the domain model for IFML). This model defines all the data types that will be used by the web application. An example domain model can be seen in Figure 3.1.

Once the domain model is defined, *site views* can be created. These site views each contain an IFML model which can make use of the domain model's data types. An example (partial) site view is shown in Figure 3.2. To keep an overview on the IFML model, it is possible to define IFML modules (see Section 2.1.3), which capture a coherent IFML model and abstract it away. Such a module is marked by the red circle in Figure 3.2.

After modelling the web application entirely in IFML, the developer can create *style projects* in WebRatio. These projects define the presentation of the IFML components that will be presented to the user. Once the developer has chosen or implemented a style project, he can entirely automatically generate the modelled web application. Finally, WebRatio allows to directly deploy the web application to a remote server.

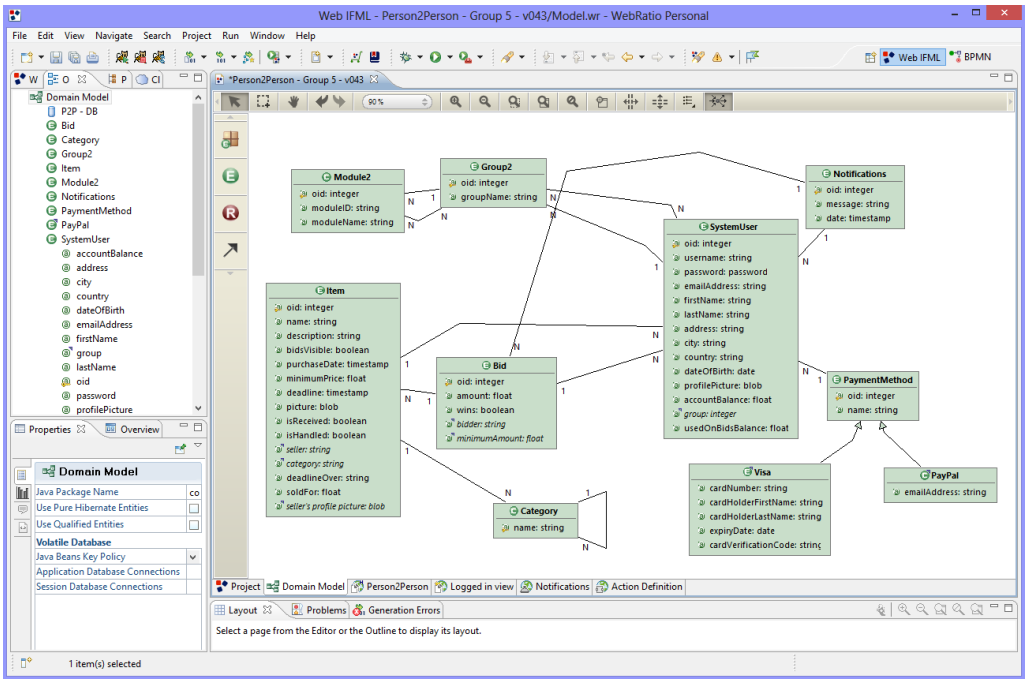


Figure 3.1: WebRatio: an example domain model

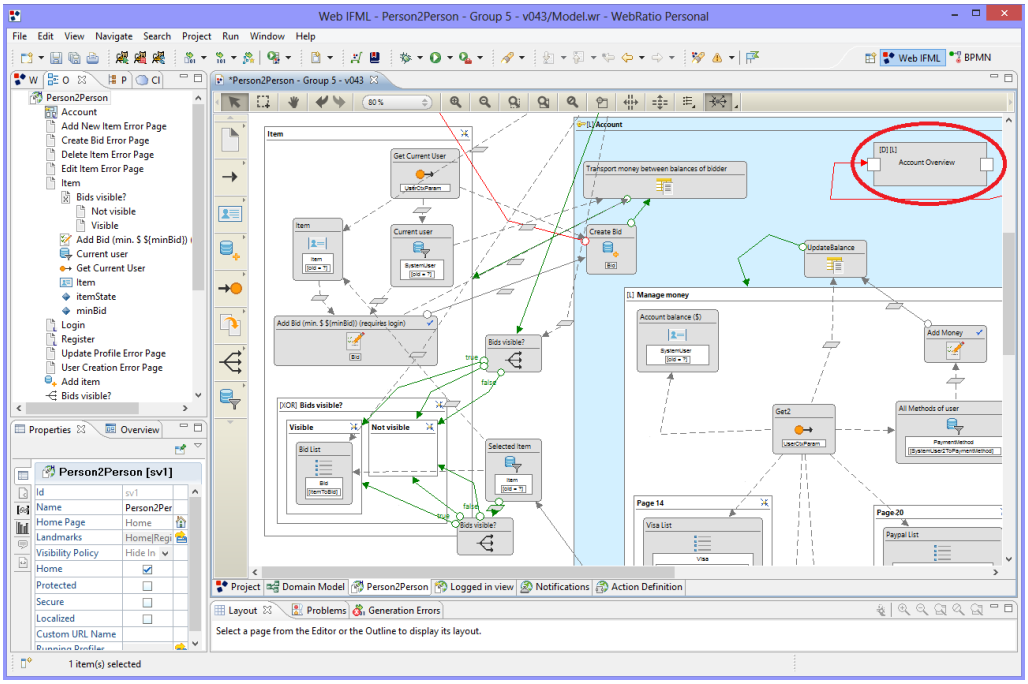


Figure 3.2: WebRatio: an example IFML diagram

3.1.2 WebRatio's IFML and Its Extensions

WebRatio allows to model site views containing diagrams which are based on IFML. The diagrams' components are not purely IFML components, even though they may seem to appear so. They are rather heavily extended to support the implementation generation WebRatio strives to provide. Figure 3.3 contains a piece of an IFML model which illustrates the extensions made to the IFML core components. This piece of IFML models a list of items owned by the user who is currently logged in. To display only the items owned by this user —and not all available items in the database—, the "My Items" list component requires an input from the "Get Username" component, which retrieves the needed username from the HTTP session parameters. This component appears as a view component, however, it is not displayed to the user and it only has a meaning at implementation level. Another component in Figure 3.3 which does not appear to the user is the "Scale Image Unit1" component. It is used to scale images to a size which the list component accepts, which makes it yet another implementation-supporting element in the IFML diagram.

Another extension to the core IFML can be found in the way WebRatio defines actions. In WebRatio, IFML actions are abstracted modules. In a

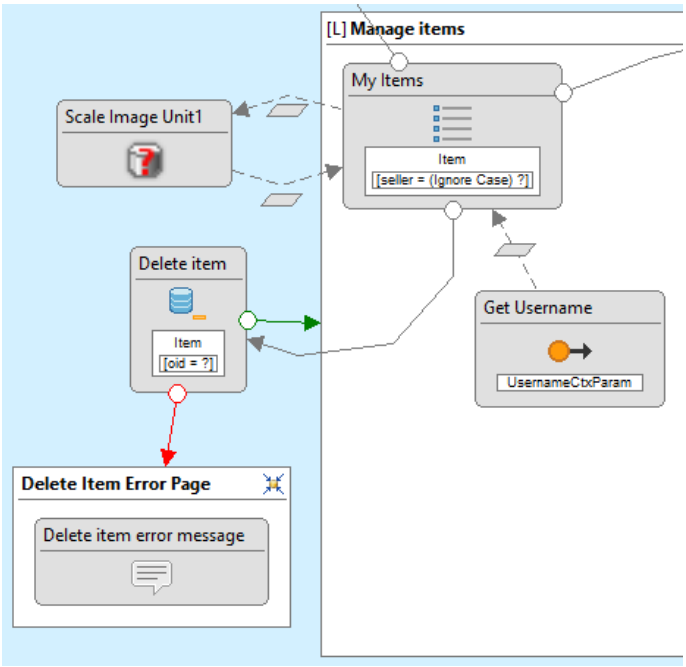


Figure 3.3: WebRatio: implementation-supporting IFML

WebRatio IFML model they keep their usual hexagonal shape, but behind that, a module with input and output port is defined to implement the behaviour of the action. In IFML, as it is defined in Section 2.1, the internal workings of an action are not elaborated. In order to generate a fully functional website however, WebRatio needs an exact specification of the IFML actions' workings. For this it uses the mentioned modules and again, as previously mentioned, uses implementation-supporting IFML to implement the specific actions. An example is depicted in Figure 3.4. Here, a log-out action is modelled with a specific WebRatio log-out view component, which represents log-out functionality rather than a user interface component.

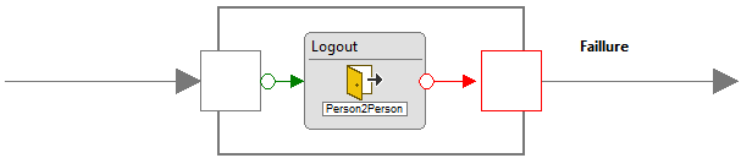


Figure 3.4: WebRatio: the implementation of an IFML action

WebRatio also allows the definition of *jobs*. These are chunks of application functionality which can be executed by the web application at a certain

time (e.g., once a day, or within 10 minutes). These too are modelled in IFML-like syntax. An example is shown in Figure 3.5. The light orange frame defines a job which performs an action on the database of the web application at a certain point in time. Even though all the components inside the job are used to perform queries on the database, they are still modelled in IFML. This demonstrates that WebRatio is rather using IFML as a general modelling language, than as a language to define interaction flow only. Most likely, this is to make WebRatio more accessible in terms of usability, as the user merely needs to get used to a single modelling language to do it all[3].

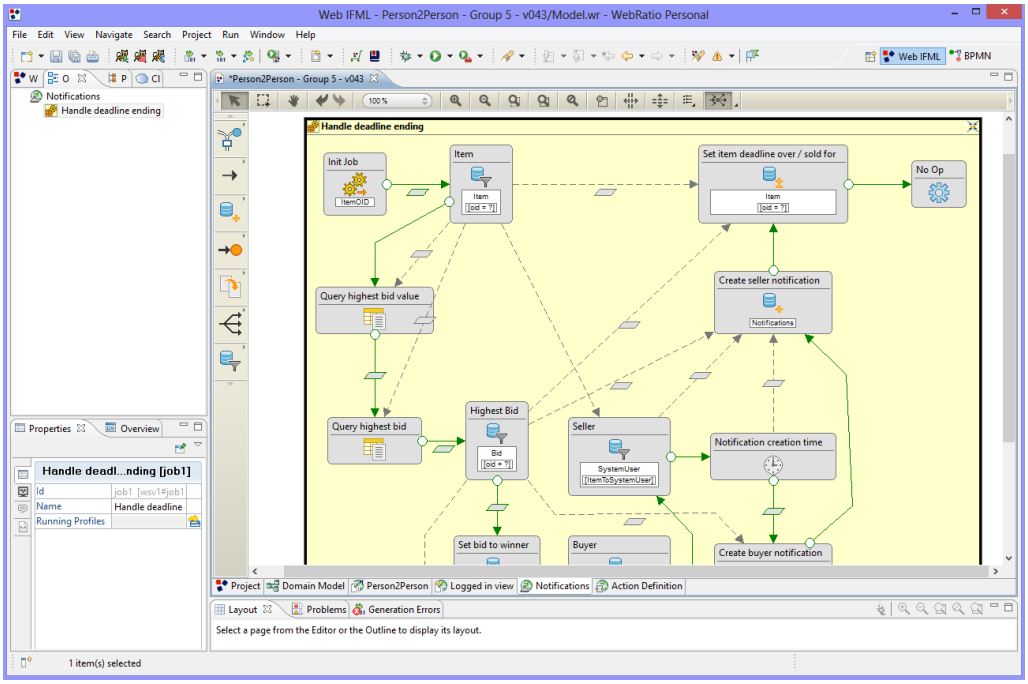


Figure 3.5: WebRatio: the implementation of jobs

3.1.3 Relation to Our Work

WebRatio as a tool supports web development with IFML from conceptual modelling through implementation. When the created IFML models are valid, WebRatio allows to generate the complete web system in one click. This resembles the adapted version of WSDM we present in this thesis as it

[3]From past experiences with WebRatio, we learnt however that it is not always straightforward to implement just any kind of functionality with the WebRatio IFML. We rather found ourselves designing overly complex diagrams to perform actions that would be easily implemented with code.

supports the same aspects of web development (not as a tool but on methodological level) and also uses IFML thoroughly to obtain a complete-system IFML model before implementation takes place. In fact, WebRatio demonstrates that it is possible to generate an entire web system from an (enhanced) IFML model. Hence, we see an opportunity for WebRatio to be integrated in the adapted WSDM to realise the step from Implementation Design to Implementation. This integration would require an extra step in the Implementation Design phase to prepare the through WSDM obtained IFML model for website generation with WebRatio. This WebRatio integration lies beyond the scope of this thesis and could be subject to future work.

3.2 Other Methods

Next to WSDM, there exist other web design methods. Some of them will be discussed briefly in this section and compared to WSDM. Note that none of these methods has IFML integrated into it.

Object-Oriented Hypermedia Design Model (OOHDM)

OOHDM (Schwabe & Rossi, 1995, 1998) is a model to design hypermedia systems which heavily relies on the object-oriented modelling approach. It proceeds in four phases: Domain Analysis (or Conceptual Modelling), Navigational Design, Abstract Interface Design, and Implementation.

During Domain Analysis, a diagram in a UML-like notation is modelled which comprises the conceptual classes in the system and the relationships amongst them. Through performing the Navigational Design phase, a layer is added on top of the conceptual classes. This layer combines parts of the classes in order to construct nodes and links between these nodes. The nodes and links can be seen as views of the user on the conceptual classes of the system. To concretise these views into user interface elements, another layer is added, this time on top of the navigational layer. This new layer contains interface classes and is created in the Abstract Interface Design phase. This concludes the design of the system. In the last phase, the Implementation phase, this design is implemented using the language and environment of choice.

As mentioned, OOHDM is an object-oriented approach and starts with an analysis of the domain to obtain classes which mainly contain the data in the system. This stands in sharp contrast with WSDM, as WSDM rather translates requirements into tasks to be performed with the system. Only after the tasks are in place, object chunks are added which specify the needed

data for each task. This difference marks the audience-driven approach of WSDM (i.e., user tasks before system objects).

In OOHDM, each phase gradually adds a new layer to the system model. This is different from WSDM in the sense that WSDM passes a model from phase to phase which is transformed in each phase. This is especially the case in the Conceptual Design phase's sub-phases: CTT task models are transformed into a navigational model which is denoted in another modelling language but nevertheless comprises all information needed from the CTTs.

OOHDM does not take into account the semantic aspects of the Web as opposed to WSDM. However, OOHDM has an enhanced successor which serves this purpose: Semantic Hypermedia Design Method (SHDM) (Lima & Schwabe, 2003).

UML-based Web Engineering (UWE)

UWE (Koch & Hennicker, 2000) is a hypermedia design method which uses UML and UML extensions as its basic building blocks. The method proceeds in four steps which subsequently create a conceptual model, a navigation space model, a navigational structure model, and a presentation model.

The conceptual model is a UML class diagram which represents the domain of the hypermedia system. The navigation space model starts with all classes from the conceptual model but links them in a different way to denote from which class to which other class the user of the system should be able to navigate. The resulting model is then transformed into the navigational structure model which elaborates on how exactly these navigational links are established between the different navigational classes (e.g., by means of an index structure). It also directly generates menus for navigation. At last the presentation model is created which describes the positioning of the navigational classes and structures within the pages of the system.

In contrast to WSDM, the UWE method clearly centralises the navigational modelling and, similarly to OOHDM, starts from a conceptual class model, due to its hypermedia nature. UWE also uses a single modelling language (UML) throughout the entire process, which is not the case in WSDM. However, it does not use standard UML only, as non-trivial UML extensions are heavily used.

A similarity between UWE and the adapted WSDM presented in this thesis is the fact that both methods use the advantage of the large expression power of their modelling languages to achieve their goal. As UWE uses UML to denote navigation and presentation models, so does WSDM use IFML to denote the conceptual navigational design of the system (whilst IFML was actually designed to focus on interaction modelling).

Object-Oriented Web Solutions (OOWS)

OOWS (Pastor, Fons, Pelechano, & Abrahao, 2006) is a UML-based web development method which also describes, next to conceptual modelling, the implementation of the web system under construction. The conceptual modelling starts with the definition of a set of diagrams which constitute the classes and objects in the system and their behaviour. Then user modelling takes place. This describes which users will use the system in the form of a user diagram. The following step is the navigational modelling. This is done at two levels of abstraction: a higher-level navigational map which defines the access of the different user types to different parts of the system, and lower-level navigational contexts which are UML packages in which is defined which data and operations the user can access when the user navigates to the context. Finally, a presentation model is created from the navigational contexts by assigning specific properties to them (e.g., a pattern for the layout of the information available in that navigational context). The implementation description of OOWS lies out of the scope of this discussion, therefore we do not elaborate on it.

Just like WSDM defines an audience class hierarchy, OOWS starts the web system modelling by defining a user diagram. Another aspect which OOWS and WSDM have in common is the hierarchical approach to navigational modelling. WSDM's navigational model has three basic levels of which one contains the audience tracks for each audience, and OOWS's navigational map defines a hierarchy of subsystems and navigational contexts, also grouped by user type.

4 Integration of IFML into WSDM

In this chapter, the investigation of the IFML (Section 2.1) integration into WSDM (Section 2.2) is presented. Before conducting this investigation, it must be recalled that the integration seeks to support the simplification and standardisation needs expressed in Section 1.2.

The investigation starts by considering alternative approaches to the integration of IFML into WSDM. Then it is continued by examining in which of the WSDM phases it could be possible to replace the currently used modelling languages by IFML. Along the way, we present our approach to integrate IFML into WSDM in great detail. Finally, at the end of this chapter, a summary of the adapted WSDM is given which sketches an overview of how the developer should carry out the adapted WSDM phases.

4.1 Integrating IFML

IFML can be integrated in two different ways. The first one obeys to the fact that IFML was created to model interaction. It can therefore only replace the conceptual modelling parts of WSDM that require interaction modelling. WSDM uses several conceptual modelling languages and notations as mentioned before. However, none of them explicitly models interaction. This

means that, when IFML would nevertheless be integrated in WSDM, it would enhance WSDM with interaction modelling facilities.

The second option is more flexible and allows to use IFML for purposes which were not anticipated by its original design. This option prioritizes the simplification of WSDM over the original purpose of IFML. Note that this resembles WebRatio's approach, as discussed in Section 3.1.2, which uses IFML more as a general modelling language rather than a language used for modelling interaction flow only.

It can be concluded that this integration investigation needs to take into account at each step whether IFML is used as it is originally defined by its specification and whether the integration is adding additional interaction modelling facilities to WSDM. Throughout this research, we will prefer using IFML as it is defined by its specification, since that is the IFML which can be understood by developers which do not have experience with WSDM, but do have experience with IFML.

4.2 Identification of the WSDM Phases That Can Use IFML

The first two WSDM phases, Mission Statement Specification and Audience Modelling, produce mostly textual output (except for the audience class hierarchy model). No extensive conceptual modelling is required in these phases and IFML should hence not be used here.

The subsequent WSDM phases, Conceptual Design and Implementation Design, both require extensive conceptual modelling. Which means there might be room for IFML in these phases. Before considering these phases in detail in the following sections, to be complete, it should be mentioned that the last phase of WSDM, the Implementation phase, cannot use IFML since it does not require any conceptual modelling.

4.3 IFML in the Conceptual Design Phase

In the Conceptual Design phase, three conceptual modelling languages are used to create the phase's output: CTT and ORM (used in the Task & Information Modelling sub-phase), and the navigational modelling language (used in the Navigational Design sub-phase). The following subsections will compare the characteristics of each of these languages with the characteristics of IFML in order to decide whether IFML can replace or partially replace the languages.

4.3.1 IFML and CTT

CTT models are focussed on how the user tasks proceed and at which moments user interaction is required. This is information that can be modelled using IFML. It is difficult however, due to the fact that the task model, in CTT or IFML, needs to be constructed from the informational and functional audience requirements. When considering the requirement "the destination address of the parcel must be entered, followed by the buyer's credit card information", which can belong to the requirements of an on-line web shop such as Amazon.com, it can be noticed that this requirement can be expressed in different ways in IFML. Figure 4.1 shows two different IFML diagrams and one CTT diagram that match the requirement. Both the presented IFML diagrams (Figures 4.1(a) and 4.1(b)) cannot result in two equal user interfaces. This means that IFML task models are in this stage yet imposing restrictions on the implementation of the web application. In CTT, this requirement cannot be specified in another way than the one shown in Figure 4.1(c). Using IFML to define task models hence results in the loss of a general conceptual task model, which a CTT tree does provide.

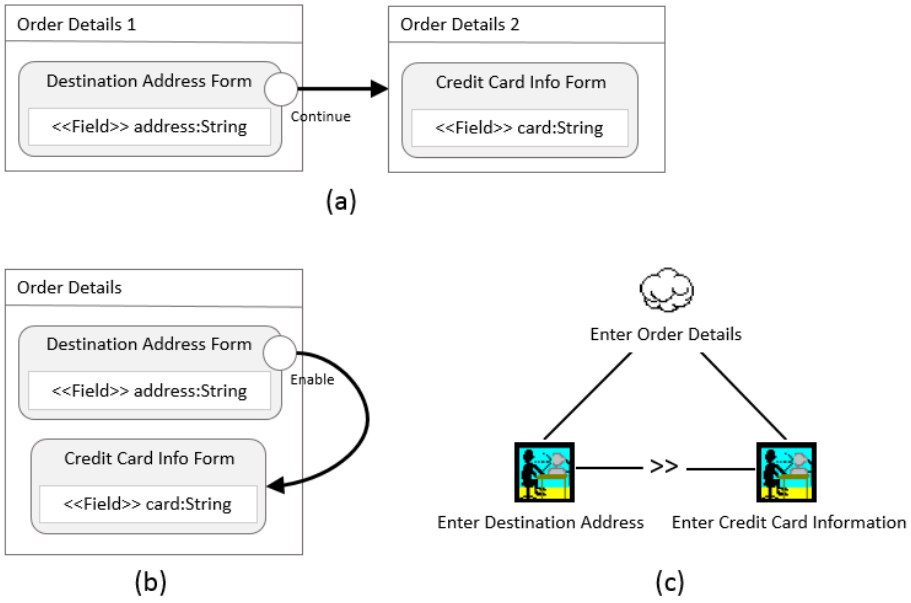


Figure 4.1: Two different IFML diagrams and one CTT diagram for the same requirement

A solution to this could be to create a more general IFML, which is a subset of the original IFML, to construct conceptual task models. The model elements of the conceptual IFML could have additional restrictions imposed

on them, to avoid situations as the one presented in Figure 4.1. For example, it can be imposed that the elements for two tasks, which are involved in a single requirement, need to reside in a separate view container unless they can be worked on concurrently or the user needs to choose between both. This would eliminate the IFML model in Figure 4.1(b) from the possibilities. To illustrate this general IFML, Figure 4.2 shows a selection of the CTT temporal operators together with their conceptual IFML equivalents. These models could then be joined, transformed and elaborated upon in subsequent sub-phases until an IFML model is created that conforms to the IFML specification. However, if IFML is used this way, certain tasks that can be easily expressed in CTT models become far more complex in conceptual IFML. This is mostly due to the fact that CTT has temporal operators to connect elementary tasks, which are much more expressive than IFML's events and navigation flows.

To illustrate the difference in complexity between both notations, the following example can be considered. Suppose an on-line flight booking application asks the user to log in with his frequent flyer account either before or after booking a flight, in order to save the earned credits for the flight on his account. Two main tasks can be identified: logging in and booking the flight. The order in which the tasks are performed does not matter. In CTT, this is expressed using the *order independent* temporal operator, as can be seen in Figure 4.3(a). In IFML, this can be expressed by the diagram in Figure 4.3(b)[1]. The latter is much more complex than the CTT model, as can be seen in the figure. It involves the use of an Interaction Flow Expression, several Activation Expressions and an abundance of arrows. This demonstrates that it can be cumbersome to directly transform a quite simple requirement in natural language into a corresponding IFML diagram. CTT models are therefore crucial to reason about tasks in a less complex manner.

Hence, since CTT task models have a higher level of abstraction than IFML task models, with immediate consequences for the complexity and understandability of the conceptual model, we opt not to interchange the CTT models for IFML models in this phase.

[1]Note that the IFML diagram is a module. This makes it possible to reuse the task where needed.

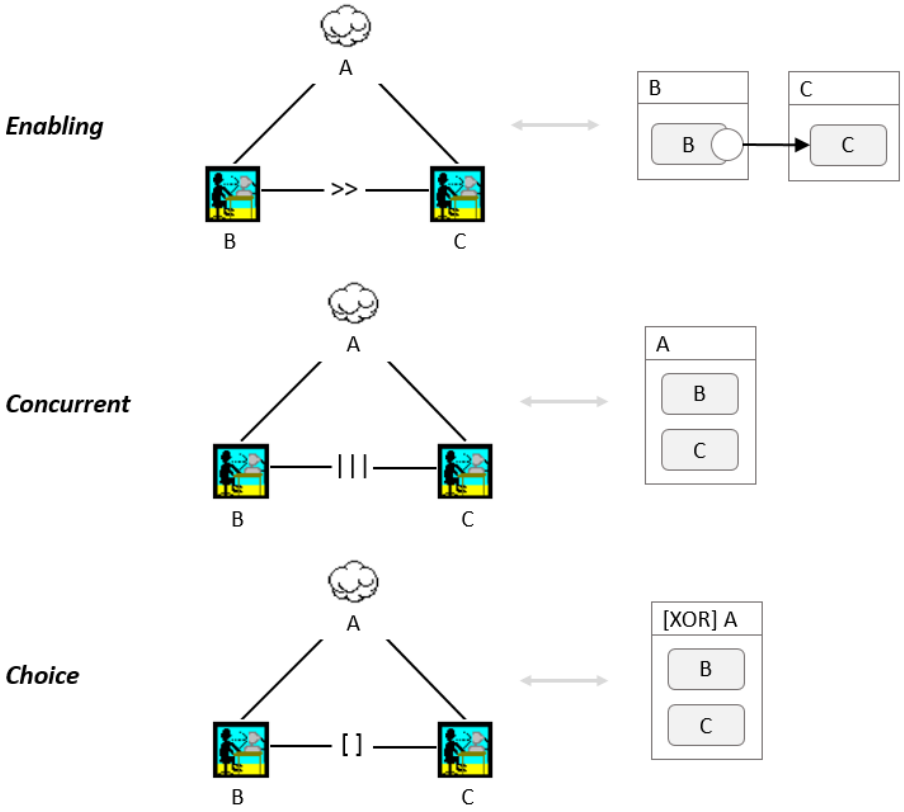


Figure 4.2: Selection of temporal operators and their IFML equivalents

4.3.2 IFML and ORM

In WSDM, ORM is used to model the information required by the CTT task models, in the form of object chunks. Since IFML is a language to model interaction flow, it is more concerned with the flow of the information than the structure of the information. IFML is not a data modelling language, hence it is not worth considering a replacement of ORM by IFML. However, we like to mention this incompatibility between both notations here, for the sake of completeness.

4.3.3 IFML and the Navigational Modelling Language

In the Navigational Design sub-phase, the CTT task models are transformed into components connected by process logic links and additional navigational aid links. Also the ORM object chunks are attached to the navigational model's components. The navigational modelling language expresses the navigational possibilities inside the web system as well as how the information

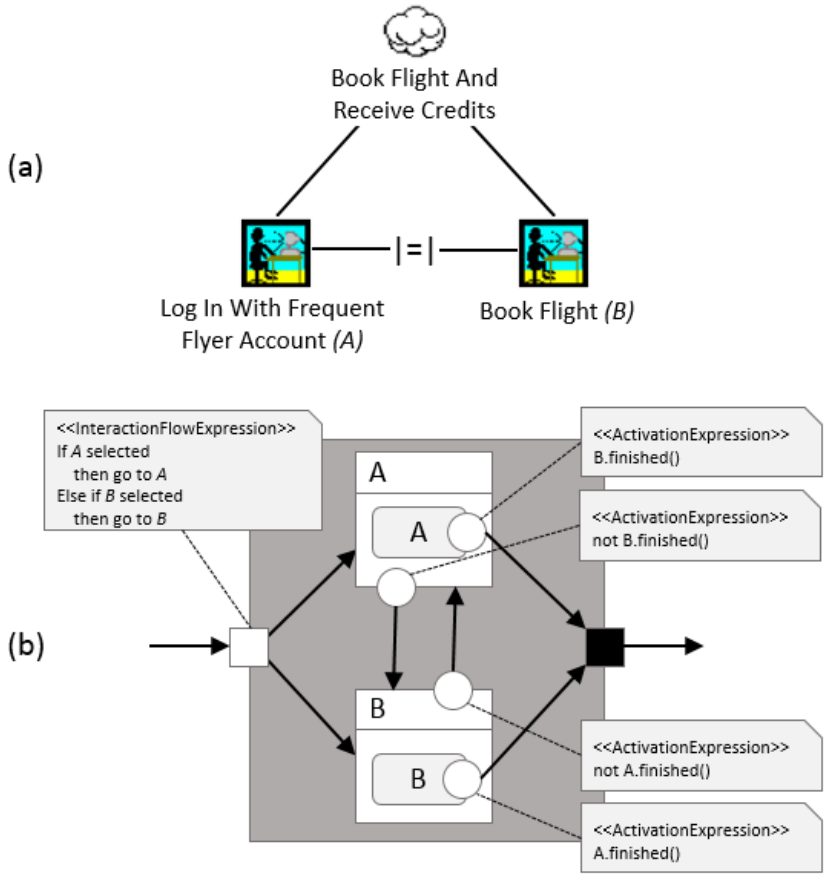


Figure 4.3: Order independent tasks expressed in equivalent CTT and IFML diagrams

flows between separate tasks. This largely matches the capabilities of IFML. However, IFML provides the conceptual model of the user interface and the interactions between the user and the interface. This aspect is not present in the navigational modelling language.

To illustrate this difference, consider the following example: a user wants to play a video which is retrieved on an on-line video sharing website. In the CTT model, this can be seen as an elementary task: “play video”. This automatically translates to a component in the navigational model. In IFML, this can result in an event attached to a video view component. Suppose there is also another elementary task for the same website, namely: “comment on video”. This would again result in a single component in the navigational modelling language. However, in IFML this would translate to more than an event only. For example, also a form for the user’s comment message

needs to be provided. It can be concluded that, in IFML, an elementary CTT task can result in different IFML constructs, which is not the case in the navigational modelling language, where every elementary CTT task is automatically transformed into a component.

IFML has direct dependencies towards the structure of the user interface. For example, IFML view containers group user interface elements, such as view components, into a compound user interface that is visible in its entirety. Hence, including IFML as it is defined by its specification in the Navigational Design sub-phase would result in a loss of conceptuality of the phase. Neither is it an option to entirely remove the Navigational Design sub-phase and use IFML as navigational modelling language in the Implementation Design phase. This would imply that there is no high-level conceptual navigational modelling done. Conceptual navigational modelling is necessary, for example, when a web system needs to be available on several platforms (e.g., desktop browser and mobile browser). All implementations could rely on a shared conceptual navigational model, which allows to add a new platform more easily since the conceptual structure is already in place.

Because the overlapping of the capabilities of IFML and the navigational modelling language is substantial, as mentioned before, it is worth considering the creation of an adapted, conceptual IFML to include in this sub-phase. This is the same approach that was considered before, when comparing IFML with CTT. For the Task Modelling sub-phase in WSDM it was opted not to replace CTT by IFML because this would result in the loss of a model abstraction level. A direct conversion from the requirements in natural language to IFML was regarded as not feasible. If the conceptual IFML would be included in the Navigational Design sub-phase, the CTTs would rather be converted into IFML instead of being replaced by it. Hence there would be no abstraction level loss.

In Section 4.3.1, a conceptual IFML was proposed. Although, at that stage it was not described thoroughly because it was immediately clear that IFML models would be too complex as first step in task modelling. For the conceptual IFML to fit in the Navigational Design sub-phase, it is necessary to define it unambiguously and in such a way that it stays purely conceptual. This is what will be done next. Because defining conceptual IFML is an in-depth and substantial process, a new section will be dedicated to it.

4.4 Definition of a Conceptual IFML

To propose a conceptual IFML that replaces the navigational modelling language in the Conceptual Design phase, a bottom-up approach will be used:

we start by presenting certain IFML constructs as alternative to the basic navigational modelling components, and then we work our way up until the highest level of the navigational model is replaced (cfr. the levels depicted in Figure 2.13).

4.4.1 Translation of the CTT Task Trees (Level 1)

The basic building blocks of the navigational modelling language are its components. Those are derived from the elementary CTT tasks that are obtained in the Task- & Information Modelling phase. The semantics of such an elementary CTT task is expressed through the task's name (in natural language) and its type (*interaction*, *application* or *abstract* task). However, when the elementary task is referred to in the navigational model, a component is created with the task's name only. Whether the task was an interaction or application task is not expressed directly, but must be deduced from the component's name and its interactions with other components.

In conceptual IFML, this can be approached differently. When a CTT task is an interaction task, it is known by definition that there must be some kind of interaction between the user and the system. We will distinguish between two main types of interaction: interaction that causes an application state change and interaction that does not. For example, when the user clicks a submit button associated to a form he just filled in, the data in the form is input in the application and hence the application's state changes. On the other hand, when the user is reading a text or watching a movie that is played back by the application, the user is interacting with the application without changing its state. The user is rather observing content in the user interface elements of the application. Strictly, this type of task would be called a *user task* in the original CTT notation presented in Paterno et al. (1997). In WSDM the notion of user tasks has been left out to avoid going in detail too much. Nevertheless, it is often natural to model these tasks as interaction tasks in WSDM. Should the developer want to do this, the IFML construction method presented here will handle these tasks correctly. Both types of interaction would be captured in a WSDM CTT task tree as interaction tasks. However, in conceptual IFML they must be assigned a different notation. State-changing user-system interaction is expressed in the form of a named IFML *event* and user-system interaction that does not imply a system state change is denoted as an IFML *view component*. To distinguish between both types of interaction tasks in the CTT model, the name of the interaction tasks that do not modify the system's state will be underlined from now on.

A CTT application task expresses that the system itself needs to do some

kind of processing that affects the user interface (e.g., updating a view). IFML has an *action* construct which allows the developer to specify that the system will execute some task —which can but does not need to have effect on the user interface. Although application tasks are more specific than actions, nevertheless can they be translated into named actions.

To be complete, it should be mentioned that CTT abstract tasks do not require any translation as they are only used to abstract away from interaction and application tasks.

Figure 4.4 summarizes the CTT to IFML transformation so far, respectively showing the transformations of interaction, application and abstract tasks. If the bottom-up approach would be continued naively, the next transformation step would be to interconnect the obtained IFML constructs according to the semantics of the temporal operators. The constructs would be interconnected with navigation flows in order to navigate from one task to the other (cfr. the navigational modelling language's process logic links). However, according to the IFML specification (Object Management Group, 2015), navigation flows can only originate from an event. In other words, it is impossible to directly connect the origin of a navigation flow to a view component or action[2]. Therefore, the latter constructs should be enhanced before connecting navigation flows to them.

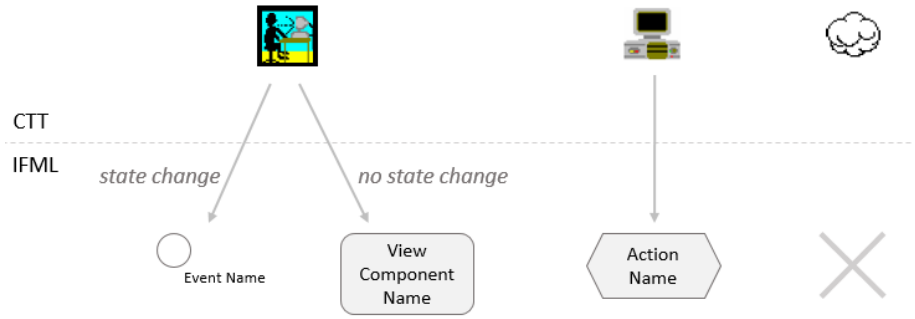


Figure 4.4: Transformation of CTT task types into Conceptual IFML constructs

In the case of actions, this can be solved easily. When an action is not a final navigation destination, i.e., when navigation needs to go on after the action has been carried out, it is good practice in IFML to attach an *action completion event* to it. This event occurs when the action is done processing

[2]Although the semantics of IFML could be adjusted to allow any construct to be the origin of a navigation flow, we prefer to comply as much as possible to the original IFML specification, as mentioned before. This prevents confusing the developer.

and allows for further navigation. Hence, any time an action is not a dead end, it should be enhanced with an action completion event.

For view components the enhancement is equivalent. When an event follows a user-system interaction which does not change the system's state, the event should be connected to the view component which represents this interaction by a navigation flow coming forth from an extra event attached to the view component. If the mentioned view component would be followed by an action, an extra event should be attached to the view component. This event is then again used as the origin of the navigation flow that leads to the action. In the case of navigating to actions, the event is typically a system event, such as a timer event which causes the attached navigation flow to be followed after a certain time.

In theory all three constructs can now be interconnected through their enhancements. There are however two more problems to cope with in order to make the interconnections meaningful. Both problems are related to events. In CTT notation, a (state-changing) interaction task can precede another (state-changing) interaction task. This would be intuitively translated into two events interconnected by a navigation flow. However, it is uncommon in IFML to navigate directly from one user-system event to another. This scenario only occurs in specific cases. For example, when a regular event occurrence leads to an asynchronous throwing event being "thrown" (and subsequently "caught" somewhere else in the interaction flow model). It is nevertheless necessary to correctly translate subsequent state-changing interaction tasks which cannot be translated as two interconnected events solely. The navigation between these interaction tasks can be seen as a transition from one user-system event to a space in which another user-system event can occur.

The second problem relates to a discussion earlier in this chapter (Section 4.3.3): an elementary CTT interaction task can be translated in either a single IFML construct or a set of constructs (e.g., a single event, or a form and an associated submit event). Thus, there must be room for other IFML components which are necessary to complete the interaction task. Specifying these components at this conceptual stage would be wrong, as they are implementation-specific.

In order to implement a solution for both the problems above, a placeholder construct can be attached to each "stand-alone" event in conceptual IFML. This construct will be formally called an *event space* and will be used as landing area for all navigation flows that would have the event associated to the event space as their destination. Later, in the WSDM Implementation Design phase, this placeholder can be concretised using 0 (e.g., in case

of an asynchronous throwing event) or more IFML constructs, depending on which facilities are needed to complete the elementary task. Important to note here is that by including this place holder, we are not extending IFML. We are merely adding a temporary construct, which will in a later phase be transformed into 0 or more standard IFML constructs. To depict event spaces, we choose for an unlabelled dashed rectangular notation that stresses their temporary nature (through the dashes) and which cannot be confused with any existing IFML construct. Figure 4.5 shows an example event space inside a course application task. The other constructs in the example are greyed out. How the process of concretisation of event spaces is carried out is subject of Section 4.5.1, further in this document.

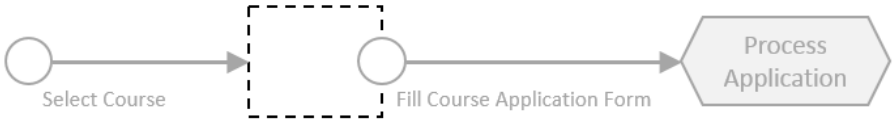


Figure 4.5: The event space representation

The IFML constructs are now enhanced to support connections between each of them. The next step is to translate the CTT temporal operators. In CTT, it is possible to have an abstract task (i.e., parent task) grouping several (2 or more) child tasks using different types of temporal operators to connect those child tasks. This introduces the "ambiguity problem" described in Paterno et al. (1997): it is uncertain which types of temporal operators should be treated first. The same paper proposes two ways to deal with this problem. A first solution imposes a total ordering of the operators which determines the order in which the tasks should be treated. In a second solution, the child tasks can only be interconnected using one specific operator while grouping the other tasks into subtasks which are abstract themselves. In the interest of the transformation from CTT to IFML, this last solution will be adopted by WSDM. Figure 4.6 shows an example of how an ambiguous tree can be transformed into an unambiguous tree using this solution. Hence, WSDM will impose that CTT task trees in which no parent has more than one type of temporal operator connecting its child tasks should be created. This will simplify the transformation process because the developer is not required to keep a total ordering in mind, and mostly because it allows to transform CTT sub-trees more systematically, as each entire sub-tree only requires a single (repeatable) transformation pattern. These transformation patterns are operator-specific and will now be explained.

Before starting the discussion of the transformation patterns, it should

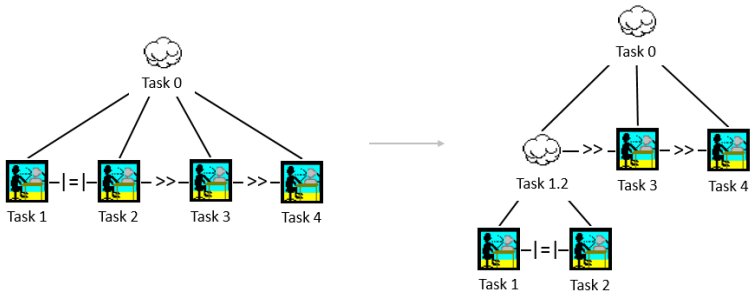


Figure 4.6: Example of how to solve the ambiguity problem according to the second method specified in Paterno et al. (1997)

be noted that in the conceptual IFML translation process directly interconnected CTT application tasks will not be transformed into several separate IFML actions. The reason for this is that CTT tasks should not model the system implementation (i.e., how the system itself carries out tasks), but rather the user's tasks. Hence, any interconnected application tasks should always be grouped in a single application task. We will not consider interconnected application tasks in the transformation patterns discussion for this reason.

Each temporal operator that connects two tasks has its own transformation pattern which translates it and its tasks into an IFML diagram chunk. Because an abstract CTT task can have two or more subtasks, the transformation patterns must be scalable in terms of the amount of subtasks the abstract task possesses. In the following paragraphs, transformation patterns for all temporal operators in WSDM are presented, together with concrete examples of their application. Note that all examples contain a CTT tree of three tasks to demonstrate the scalability to more than two tasks.

Enabling (>>) and *Enabling with Information Exchange* ([]>>)

The transformation pattern of the enabling operator is trivial. One task needs to be performed after the other one finishes. When there is information exchange between the two tasks, an IFML parameter binding should be added to the navigation flow. The transformation patterns are shown in Figure 4.7. Note that this figure only presents the case of a state-changing interaction task following an interaction that does not change the application's state, and an application task. Other task type combinations are possible too. Their mapping is however equivalent to the presented example and is not shown to not overload the reader with all possible diagrams. This is the approach we take for the discussion of all temporal operators.

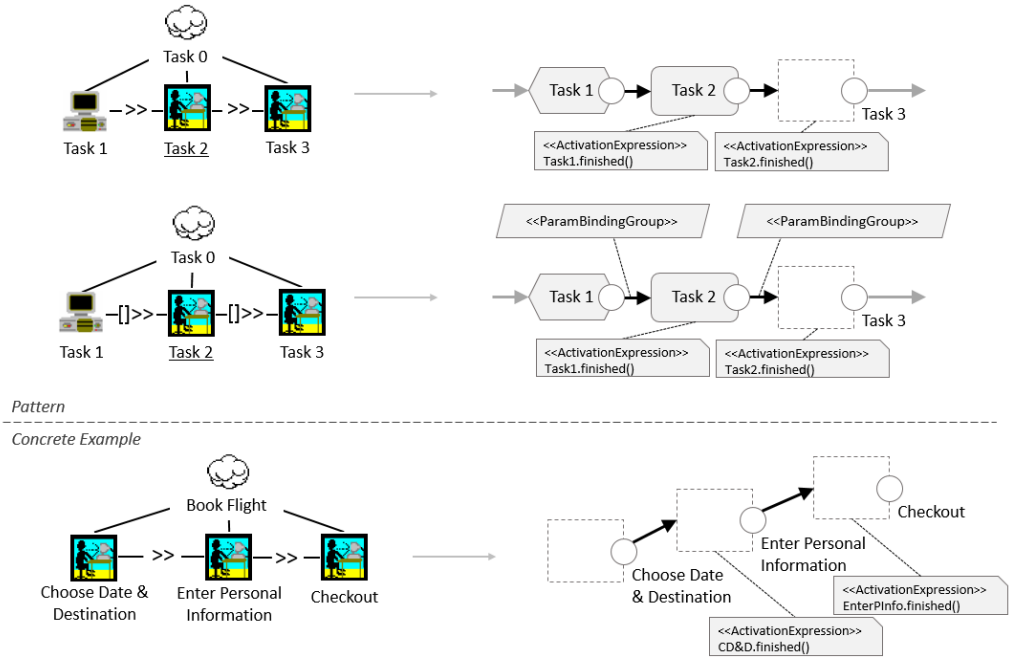


Figure 4.7: Enabling (>>) and Enabling with Information Exchange ([]>>) transformation patterns

The activation expressions used in this pattern are to guard the fact that the user cannot access any other step than the one he currently needs to execute. Those expressions are needed, since it is possible that the tasks will be grouped on a single page in a later phase of WSDM. If this would be the case, the sequential constraint specified in the task modelling phase cannot be violated because of the activation expressions.

Suspend-Resume (|>)

The left-hand task of this operator can be put on hold to execute the task on the right-hand side of the operator. Once the latter is completed, the suspended task is continued. Figure 4.8 shows the transformation pattern of this operator. An activation expression is again used to guard the user from accessing a not-yet-available task. Note that we deliberately did not use an application task for the example figure of this temporal operator because application tasks are mostly executed as a whole. Neither is it possible to interrupt an action in IFML to continue it on a later moment in time. Actions in IFML can only have two types of events attached, as described in Brambilla and Fraternali (2014): normal termination (also known as an action completion event), or exceptional termination. It is hence not possible

to interrupt an action. It can only be terminated.

The concrete example given in Figure 4.8 was inspired by an example given in Paterno (2003). Printing temporarily suspends working on a document. When printing is started, the user can continue editing the document.

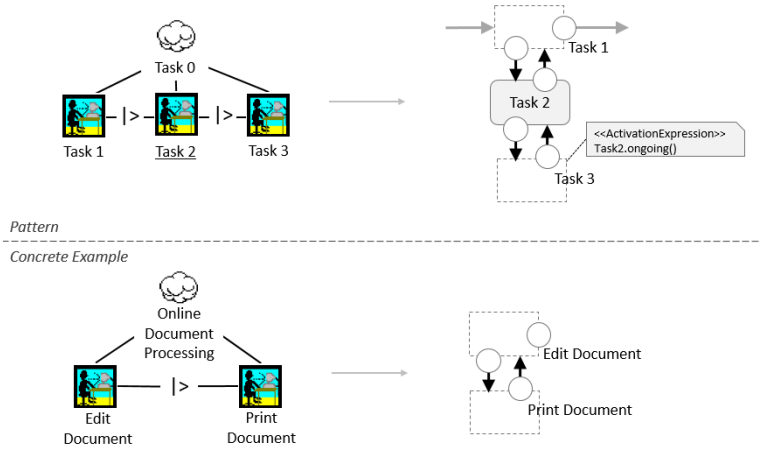


Figure 4.8: Suspend-Resume (|>) transformation pattern

Deactivation ([>)

This operator expresses that a first task is deactivated upon starting a second task. Note that the deactivation operator cannot have an non-state-changing interaction task as its second task because there needs to be state-changing interaction with the system to be able to end a task in progress. Also an application task can be the deactivating task (e.g., using a system timer that allows the user to execute the first task for only a certain amount of time). However, an application task itself cannot be deactivated. As mentioned in the Suspend-Resume operator discussion, application tasks can only terminate normally, or be terminated by an exception.

Figure 4.9 shows the transformation pattern of this operator. Notice that an extra Deactivate event is added to the event space of Task 2. This event needs to be there because actions cannot start up themselves. For example, it can be a system timer event, as previously mentioned. Task 2 itself is not the triggering event in this case. It can nevertheless be completed yet before the deactivation event occurs. An activation expression is attached to the Task 2 event to express that the event can only occur once. Without including this expression this event could occur infinitely many times because it does not have a navigation flow to follow.

In the same figure, a concrete example is given which presents an online

examination system in which the user needs to answer questions within a 10 minutes time limit. Once this limit is reached, a system timer event activates an action which terminates the editing of the exam.

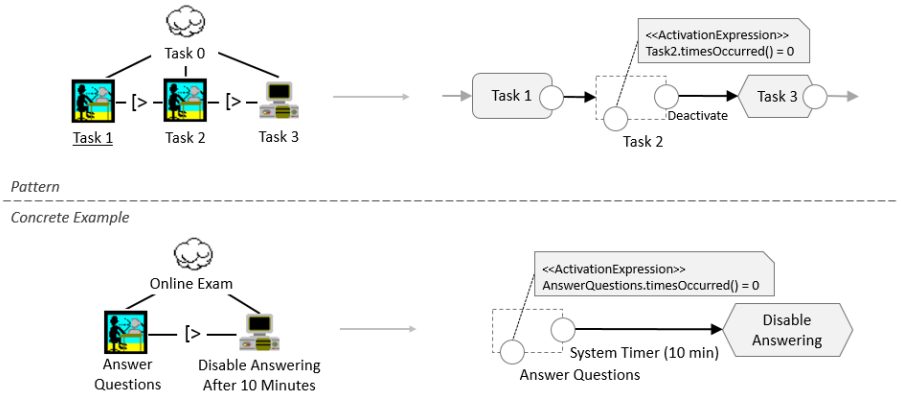


Figure 4.9: Deactivation ([>) transformation pattern

Interleaving (|||) and *Interleaving with Information Exchange* (|[]|)

Tasks connected with these operators can be executed in an interleaving manner. The user can work on both tasks at the same time, switching from one to the other as many times as is needed. The translation of this operator depends heavily on the task types of the tasks it connects. As can be seen on Figure 4.10, whenever an application task is present, it should always be started at the same time the user is able to start performing the interaction tasks. Therefore, the translated action is initiated through one of the event spaces or view components it is concurrently executed with. To achieve this, a specific event is used which directly triggers the action at the time the associated event space or view component is activated. Because we cannot express in standard IFML that this event should be triggered upon arriving in the related event space, we give it a meaningful name, OnActive, which expresses this requirement. Notice also the activation expression attached to the Task 2 event. It again guards the fact that the task can only be executed once. To prevent that one task is left unfinished when another task is completed, it is necessary to attach an extra event to explicitly leave the event space. This event can only occur when all state-changing interaction events are completed and is not part of the transformation as long as there is no subsequent functionality. Therefore it is greyed out in the figure.

In the case of concurrency with information exchange, data flows and extra parameter binding groups need to be added in order to exchange infor-

mation with the action. Information exchange is implicit between the view component and the event space because the first resides in the latter.

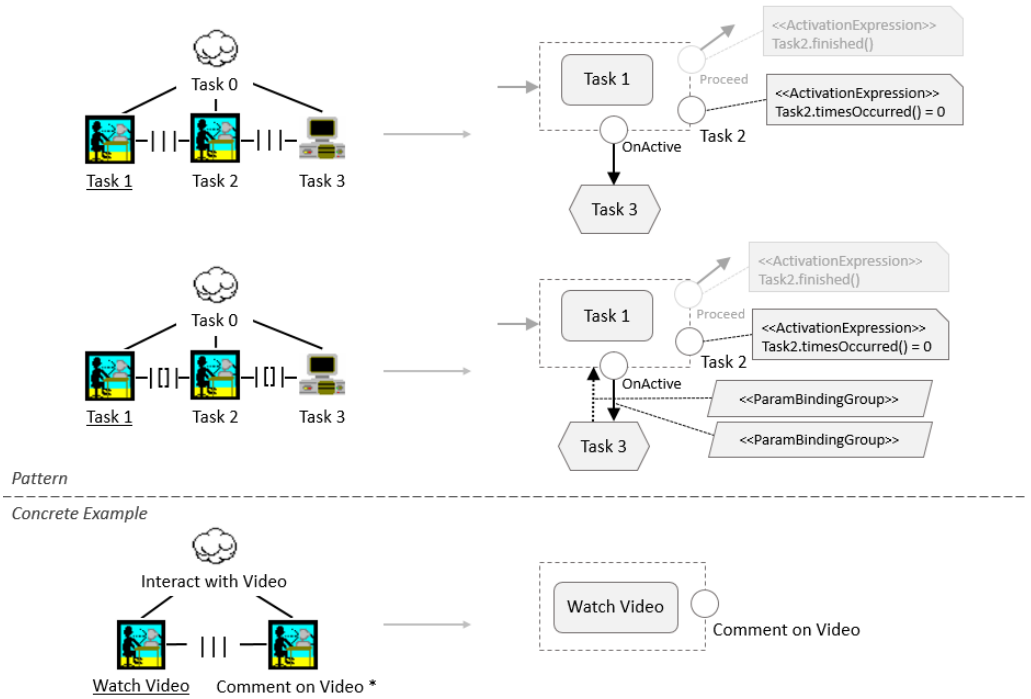


Figure 4.10: Interleaving (|||) and Interleaving with Information Exchange (|[]|) transformation patterns

The terms *Interleaving (with Information Exchange)* and *Concurrent (with Information Exchange)* are alternatively used to refer to this task type. The original WSDM chapter (De Troyer et al., 2008) uses the latter term. However, we believe "interleaving" is more appropriate because "concurrency" is contra-intuitive in describing system interaction as a user cannot perform two actions at the exact same time. The user rather interleaves completing pieces of each task to finally complete all tasks.

Note that in the case of the patterns in Figure 4.10, the temporal operators can also be translated by adding navigation flows to both the action and the event space separately, starting from a single preceding event. However, since such an event is not always available (e.g., when there is no preceding functionality), we opted for the translation presented in the figure.

Important to mention is the following. When two or more state-changing interaction tasks are present in the interleaving CTT tree, they are translated to a single event space with an event for each task residing on the edge of the event space. Also, when two or more view components are present, without

any event space, they are merged into one view component with as name the aggregation of all task names, linking them with the symbol "+". In a later phase the view components can be split up again if necessary. In the case of an aggregated view component, any accompanying action is attached directly to the view component.

The concrete example in Figure 4.10 shows the transformation to an IFML model which models interaction with a video by watching it and by commenting on it. Important to note is that in this example, there is no need to limit the amount of occurrences of the video commenting event since it is iterative by definition in the CTT tree, denoted by the star (*). It can hence be repeated as many times as desired. More on this iterative operator will follow soon.

Choice ([])

This operator forces the user of the application to proceed with one of the tasks that are available to him. The other presented tasks are no longer considered after the user makes his choice. In Figure 4.11, a transformation pattern for this operator is presented. When the three tasks do not have preceding functionality, they float around separately, not interconnected, in the conceptual IFML model. However, an IFML action should always have a preceding trigger, denoted by the arrow in black in the figure, because it cannot start up itself by definition. To assure that in no case more than one of the three tasks is performed and that none is performed more than once, activation expressions were assigned to each of them. Note that an action cannot have an activation expression. Therefore the expression is assigned to its triggering event which is always present. Even when the tasks are preceded by other functionality, and are thus navigated to by the greyed-out navigation flows separately, it is still mandatory to include all activation expressions. This is because the tasks might still be grouped, in a later phase, in a single component in the user interface which is visible in its entirety. This would make them accessible at the same time.

As concrete example of the choice operator, Figure 4.11 depicts a CTT tree which lets the user choose between creating a seller or a buyer account on an online auction platform. Once a seller account is being created, it is no longer possible to create a buyer account. This also holds the other way around.

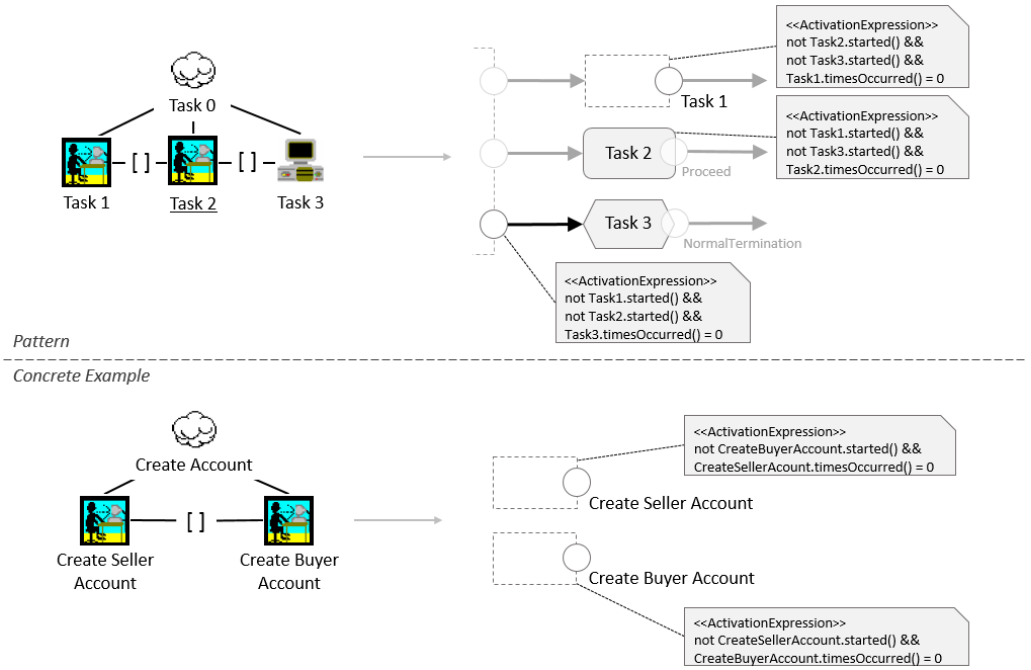


Figure 4.11: Choice ([]) transformation pattern

Order Independent (|=|)

Tasks that are interconnected with this temporal operator can be executed in whichever order the user wants to execute them. There is however one constraint: all tasks need to be executed. Because application tasks should not be part of order independent groupings, as they can typically be executed before or after the order independent group, they are not included in the transformation pattern which is shown in Figure 4.12. Once again, activation expressions are extensively used to guard the occurrence of events. Note that in the transformation pattern shown in the figure, only the four activation expressions that are associated with Task 1 are depicted. Both Task 2 and 3 also need equivalent activation expressions. They are however omitted to not overload the figure.

To proceed to the subsequent functionality in the system, the three tasks have special activation expressions assigned to their Proceed events which prevents it to be enabled when not all of the tasks are completed. As can be seen on the figure, all proceed events lead to the same event space. This is due to the definition of the Order Independent temporal operator: whichever order is chosen, the user arrives, after completing all tasks, at the same place in the user interface.

The concrete example for this operator models a system to sell goods on

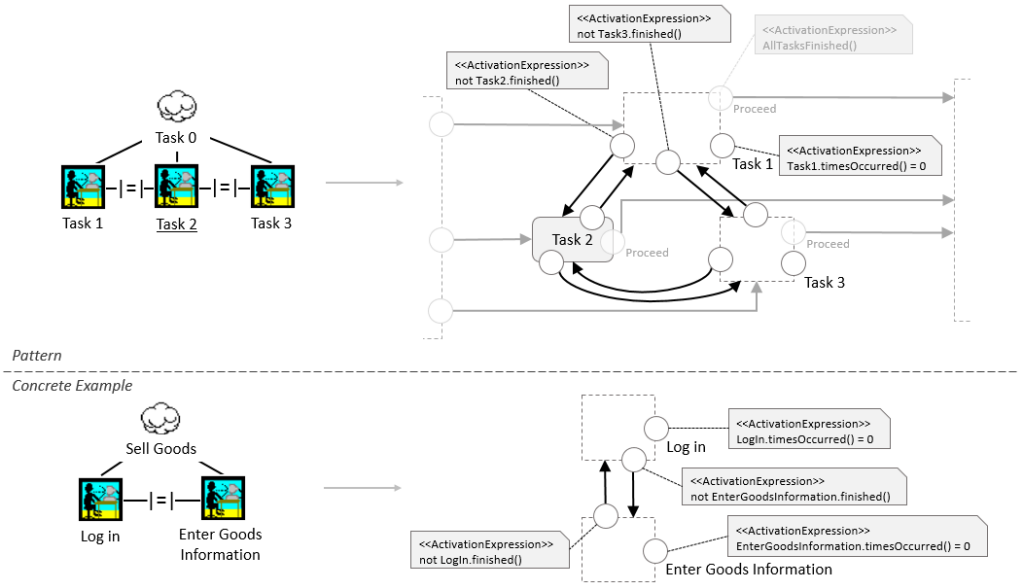


Figure 4.12: Order Independent (|=|) transformation pattern

an online auction platform. The selling users are allowed to log in either before or after entering information about the goods they seek to sell. The order of both tasks is independent.

Iteration (Task*), *Finite Iteration* (Task(n)) and *Optional* ([Task])

These temporal operators are, unlike the ones specified so far, unary operators. They add meaning to one CTT task only. These three operators are grouped together in a single paragraph because they are closely related. *Iteration* specifies that a task should be performed once and can then be repeated as many times as desired. *Finite iteration* imposes that a task should be repeated exactly n times and *optional* tasks do not necessarily need to be performed. It is also possible to combine these operators. For example, a task can be iterated and optional. This implies that it can be performed as many times as wanted, but does not need to be performed even once. Each of these temporal operators has some control over how many times a task is executed.

It can be observed that there is no control over the number of times a non-state-changing interaction task is performed. For example, the user can (visually) look up a record in a list that is presented by the system. There are however no limitations to how many times a user wants to repeat this, as this interaction is not part of the system and cannot be tracked. It is thus not meaningful to mark this type of interaction task as iterative or finitely

iterative in a CTT model. The meaning of optional non-state-changing interaction tasks can be questioned too. If there is some user interface component available to perform the interaction, it is for the user to decide whether he wants to perform it or not (e.g., pick a record from a list with or without analysing the list). The system has no means to check if it was performed. Therefore, it is not meaningful to translate optional characteristics of non-state-changing interaction tasks to IFML.

Also application tasks are not directly compatible with iteration. An iterated application task represents the system carrying out a task several times. This would be modelling the system's implementation, which is not part of the task modelling phase. The iterations in the system's implementation should rather be captured within the application task. This reasoning expands to optional application tasks. When an application task is optional, it is either performed or not. However, if this decision is the responsibility of the user, the application task would be preceded by some interaction task which represents the user making his choice. In this case, the abstract task which comprises both the interaction and the application task is optional, but not the application task itself. If on the other hand the decision is to be made by the system, its implementation would again be modelled in the CTT tree, which is not desired.

It can be concluded that only the iterative and optional characteristics of state-changing interaction tasks and abstract tasks can be translated meaningfully into IFML constructs. Because abstract tasks will be subject further on in the conceptual IFML discussion, we focus on state-changing interaction tasks here.

Figure 4.13 presents for each of the unary operators the adjustments that should be made to an IFML model of a state-changing interaction task in order to support the operator. For all operators, if their underlying task is not a dead end, an extra event needs to be added to the event space to allow the user to proceed in the application. Through a combination of activation expressions attached to both events, the operator is translated. For example, finite iteration is expressed by limiting the number of occurrences of the Task event to n and only activating the proceed event upon the nth occurrence of the Task event.

An iterative or optional state-changing interaction task is, unless it is the only task in the CTT tree[3], always part of a non-unary temporal operator, such as those seen previously. When this temporal operator is translated yet, it is often possible to only adjust the IFML slightly in order to support

[3]If the task is the only task in the CTT tree, then it should be translated as in Figure 4.13.

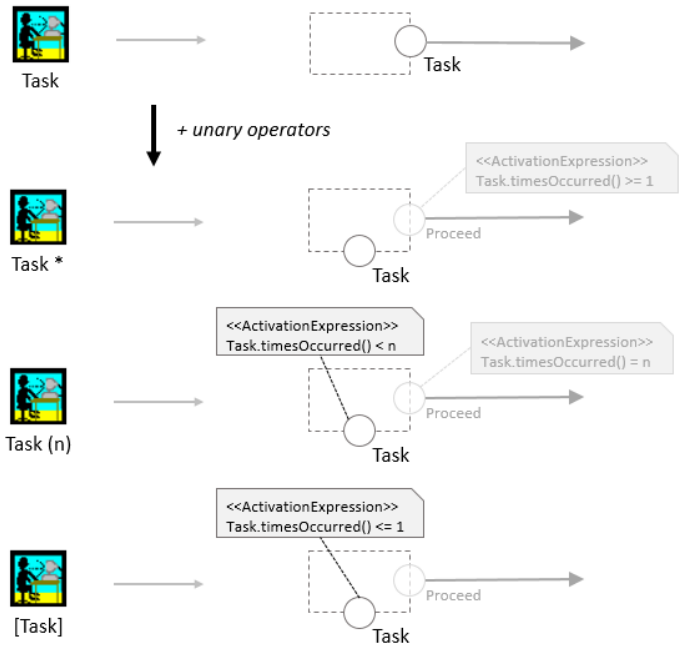


Figure 4.13: Iteration (Task*), Finite Iteration (Task(n)) and Optional ([Task]) transformation patterns

iteration and optionality because the translation of the non-unary operator already partially covers the needed adjustments. Figure 4.14 illustrates this. The figure depicts the adjustments made to the translation of an interleaving CTT tree. In this case only adjustments to the activation expressions of the translated operator need to be made. The differences are marked in red.

Note that the concrete example in Figure 4.10 covers an practical example of the translation of an iterated task.

Recursion

Recursive tasks make it possible to go back up in the CTT tree to start an abstract task tree over again, by reusing the name of the abstract task[4]. In this case, an abstract task can be a leaf of the CTT tree. To facilitate recursive tasks, navigation flows can be used to navigate back to the root of the recursive sub-tree. Figure 4.15 shows an example. The presented CTT tree models a system to compose a development team of managers and supervised employees. To translate it, the recursive tasks should be ignored at first. Consequentially, the two abstract tasks “Add Manager” and “Add

[4]In CTT notation, the task’s name uniquely identifies the task.

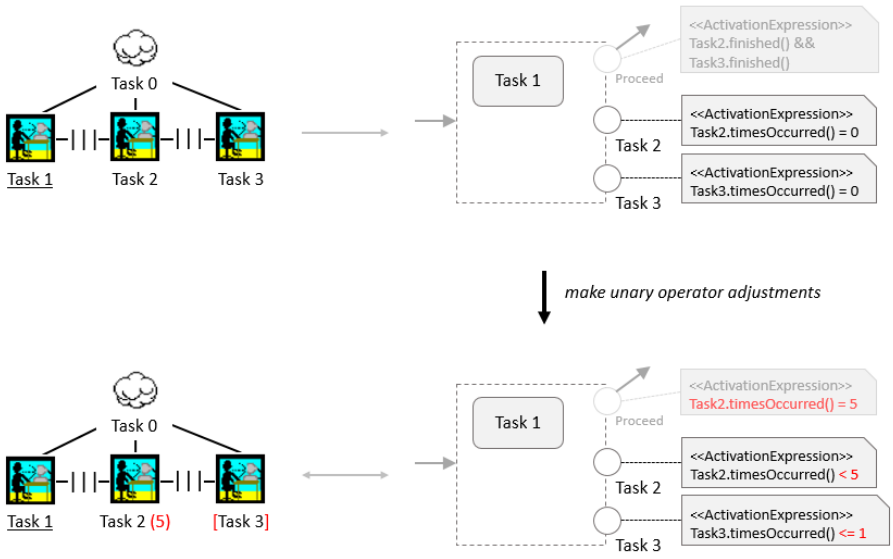


Figure 4.14: Unary adjustments to the translation of a non-unary operator

Employee" result in two events with associated event spaces. When the choice operator is then translated, an extra "Submit Team" event and event space are added, together with three activation expressions (in accordance with the choice transformation pattern in Figure 4.11). Finally, to support the recursive tasks, extra constructs are added and existing ones slightly adjusted. Those constructs are marked in red.

Recursive events with navigation flows are added to flow back to the root of the task tree. Due to the fact that the choice operator is at the base of this tree, each event space which supports recursion must have a navigation flow connecting to each other event space. For the model to be transparent and consistent, each recursive event will have the "Recur:" prefix added to its name. Note that there is no recursive navigation flow from an event space to itself. This is because the event corresponding to that event space can be executed as many times as wanted. Hence it already supports the recursive aspect and does not need an extra flow.

The task tree models enabling tasks with information exchange. The reason for this is that the partially constructed team must be carried to the next recursion cycle to be able to attach a new employee to an already existing supervising manager. To support this information exchange, parameter bindings need to be added to each recursive navigation flow, as can be seen in the figure. To make the transformation complete, the activation expressions which support the translation of the choice operator should be adjusted to conform to the recursive nature of the task tree. Instead of enabling the event

spaces only when none of the tasks was ever started, they are now enabled when no other choice task is currently ongoing. This is to allow recursion to reactivate the event spaces.

The approach taken here can be easily mapped to other temporal operators. For each temporal operator described previously, incoming flows were shown (in light gray) in their corresponding transformation pattern figures. These flows should be taken into account when applying the recursive transformation pattern to other temporal operators, as the recursive tasks reconsider the recursive sub-tree through these entry points.

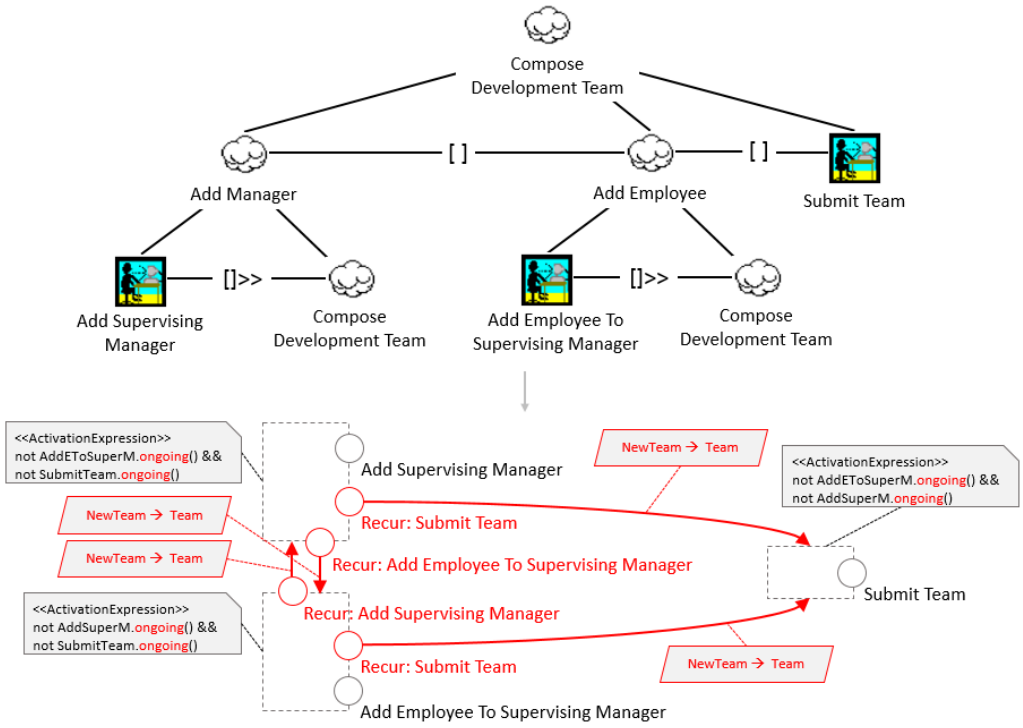


Figure 4.15: Recursion transformation pattern

Transaction (->Task <-)

When a task is enhanced with the transaction operator, it has the ability to have its state-modifying actions rolled back if one of its subtasks fails or is aborted. This operator was originally included in WSDM. However, we decided not to support it here, because it reflects the reaction of the system to actions taken by the user rather than describing the tasks the user is able to perform.

Up until now, a means to transform any CTT tree of depth 1 which is based on a single temporal operator was given. Now, an algorithm which uses the described transformations to translate CTT trees of any depth will be presented.

The algorithm processes the to-be-translated task tree bottom-to-top: first the sub-trees which only contain leaf tasks will be translated using the seen temporal operator transformation patterns. The yielded results will then be used as input for the translation of the tree level in which the respective parent nodes of the leaf nodes reside. This mechanism is illustrated in Figure 4.16. The tasks which are marked by the red frames are to be translated first. Note that in the first step of this example two sub-trees at a different level in the tree are translated. Once this is done, the yet-translated abstract tasks, denoted by the green border around their icon, are used as a whole in the translation of the next level in task tree. This is repeated until the root of the tree is reached.

The first step in the algorithm is covered by the transformations explained earlier. The subsequent steps, which treat the higher levels of the CTT tree, are largely based on these same transformation patterns, however, they now also need to support abstract tasks. These abstract tasks are already translated into IFML diagrams in the previous step(s) and hence need to be fit into the current translation rather than be translated themselves. How this can be done will be explained in the following paragraphs.

To have a formal basis to support the translation of abstract tasks over several levels in the CTT tree, the term *enabled task set* (ETS) (Paterno, 2000) should be introduced. An enabled task set is a set of tasks which the user can access at the same time. For example, when the user has a choice (through a choice operator) to carry out one of three available tasks, these three tasks are the enabled tasks in that user's current enabled task set, because he can start performing either one of them. In the discussion at hand, two enabled task sets in specific are needed: the initial enabled task set of the abstract task and its final enabled task set. The tasks in these two sets are the only ones of which the translation will be connected to the translations of the tasks which directly precede, follow or interleave the abstract task under consideration. These latter tasks will be called the *surrounding tasks* of the abstract task.

Paterno (2000) defines a function `first(Task)` which returns the initial enabled tasks of a given abstract task. This function can hence be used directly here. To determine the final enabled task set, a `last(Task)` function should be introduced. This function returns each task that is (possibly) the last task which is performed to complete the abstract task. In Figure 4.17,

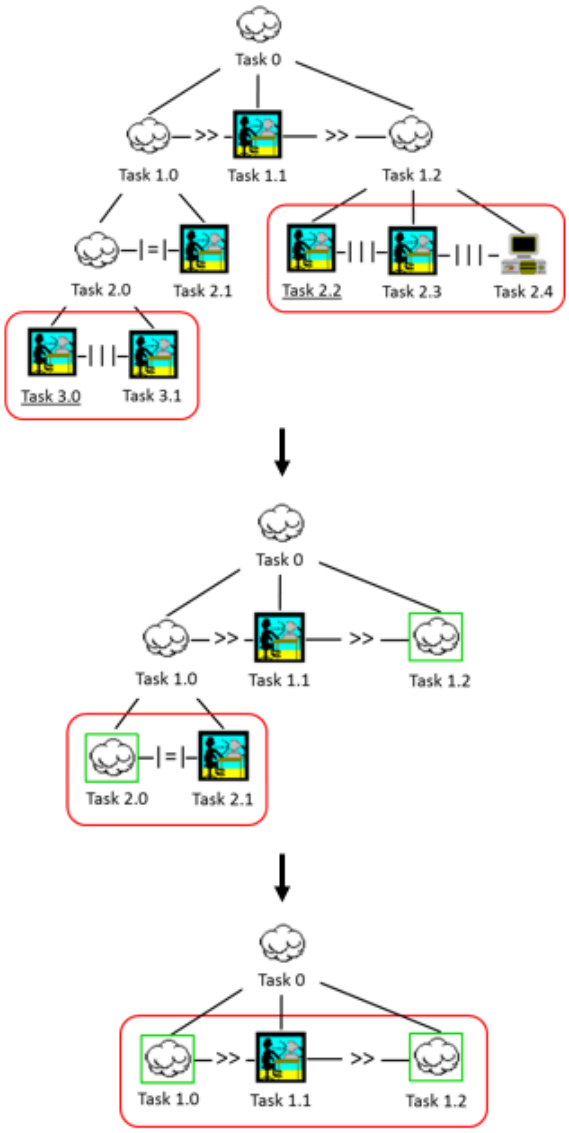


Figure 4.16: Working of the task tree translation algorithm

an overview of the discussed terminology is presented in an example which models a paper submission system. Due to the order independent operator, all child tasks of the Paper Submission task are available at the same time. Hence, the surrounding tasks of the Choose Journal task are both other order independent tasks.

Because the enabled task sets differ for each temporal operator, the following overview, which is partially based on Paterno (2000), is presented.

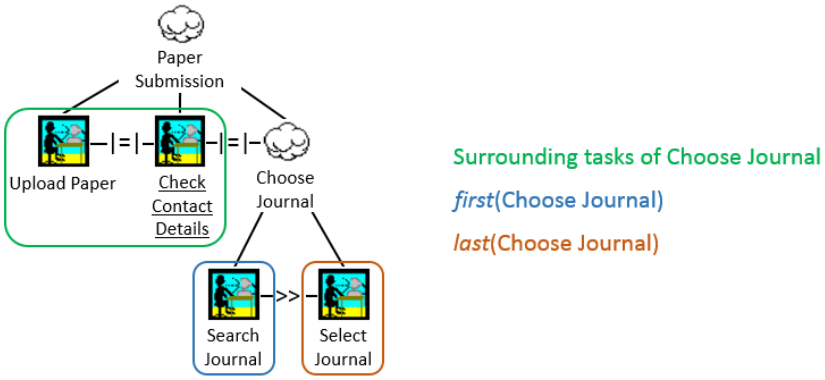


Figure 4.17: Task tree terminology

It lists the tasks in the initial and final enabled task sets of each temporal operator[5]:

- Enabling (>>):
  - Initial enabled task set: The first task in the sequence
  - Final enabled task set: The last task in the sequence
- Suspend-Resume (|>):
  - Initial enabled task set: The first two tasks in the sequence
  - Final enabled task set: The first task in the sequence
- Deactivation ([>):
  - Initial enabled task set: The first two tasks in the sequence
  - Final enabled task set: The last task in the sequence
- Interleaving (|||):
  - Initial enabled task set: All tasks
  - Final enabled task set: All tasks
- Choice ([]):
  - Initial enabled task set: All tasks
  - Final enabled task set: All tasks

[5]Note that when a task in an enabled task set is abstract, the `first` or `last` function should recursively be applied on this task to obtain the initial and final task sets.

- Order Independent (|=|):
  - Initial enabled task set: All tasks
  - Final enabled task set: All tasks

The tasks in the initial enabled task set will be the entry points of the abstract task to which its surrounding tasks can pass control. Once the abstract task is performed, the corresponding final IFML components (the exit points defined by the final enabled task set) pass control back to the surrounding tasks. Note that in the transformation patterns of the temporal operators depicted earlier, both the entry points and the exit points were already denoted by their connected light gray navigation flows. These navigation flows hence navigate from and to the surrounding tasks of the abstract task. An example is shown in Figure 4.18. This example presents the translation of the paper submission system. In the figure, the red rectangle denotes the parts in the translation which correspond to the enabling tasks. The green navigation flows in the translated IFML diagram represent the entry and exit flows between the abstract task and its surrounding tasks.

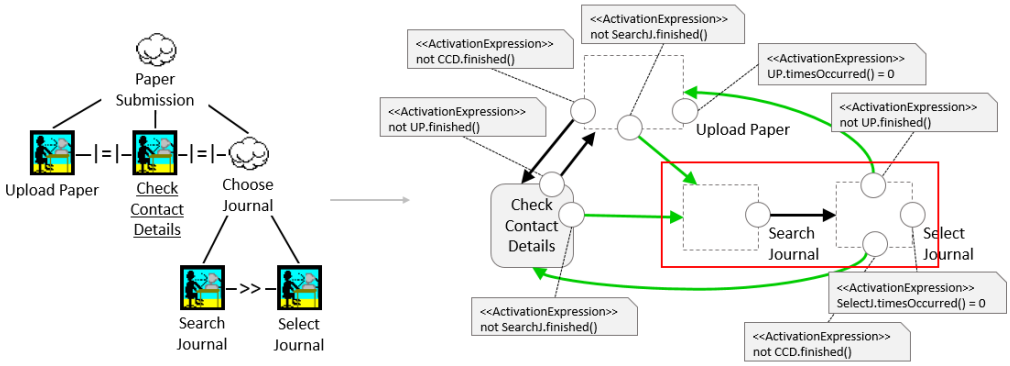


Figure 4.18: Order Independent - Enabling translation combination

This reasoning applies not only to the order independent and enabling combination but also to other operator combinations. Every combination of two temporal operators of which one is present in a sub-tree of the other has its own specific translation. Describing and depicting each combination separately would make this document too large, taking into account that there are about 36 combinations of the 6 non-unary operators. It is however not impossible to do this in future work and hence it is still possible to automate the translation process.

However, to give the reader some insight in the translation differences amongst the operator combinations, a few more combinations will be dis-

cussed. A first combination involves the deactivation operator and the suspend-resume operator. Figure 4.19 depicts part of an online conceptual modelling tool. The editing of the model can be paused to search through the help index. This task on its turn can be suspended to print the single help topic that was found. At any point, the user can choose to close the model he is working on. To support the latter, deactivation events which reside at each of the suspend-resume tasks connect to the deactivating task using navigation flows. It can be concluded that whenever a deactivating task is translated, all subtasks in the deactivated abstract task must connect to it. This is a special characteristic which also counts for the suspend-resume operator, since by definition it can pause the suspended abstract tree at any subtask. In this specific combination it is hence not needed to take the initial and final enabled task sets into account, as the translation of the surrounding disabling task does not rely the initial and final tasks of the suspend-resume task set.

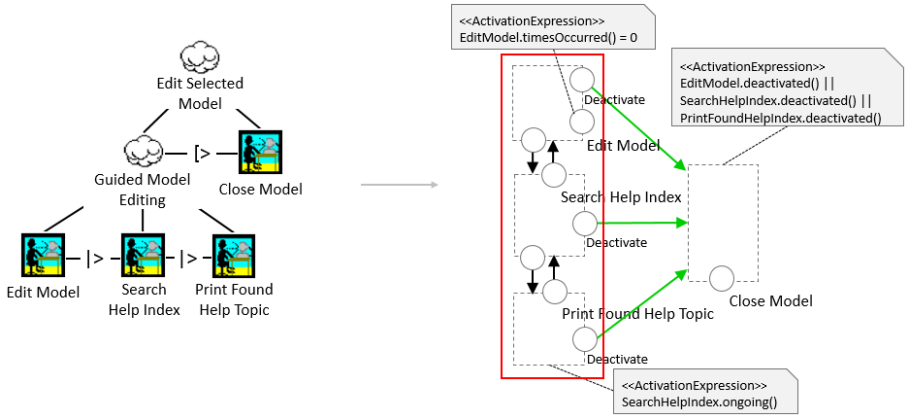


Figure 4.19: Deactivation - Suspend-Resume translation combination

In order to show that the algorithm also works for trees with a depth larger than 2, the next example, which models a system to monitor, buy and sell stock, presents a task tree with depth 3 and contains the following two operator combinations: Interleaving - Choice, and Choice - Enabling with Information Exchange. Note that it also involves the iteration of an abstract task. To demonstrate the working of the algorithm in a step-by-step manner, each algorithmic step is presented separately in Figures 4.20 through 4.22. In the first step of the algorithm the sub-trees which only contain leaf tasks are translated. In this case, the enabling sub-tree.

The second step, depicted in Figure 4.21, takes the output of step 1 and adds the translation of the choice operator to it. This results in an additional event space for the Sell Stock task and an extra activation expression for the

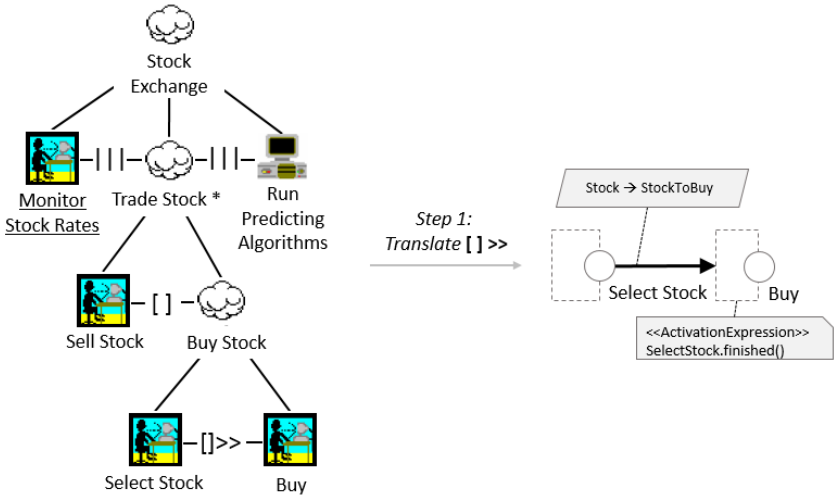


Figure 4.20: Step 1: Enabling translation

Select Stock task, to prevent it from being performed concurrently with the Sell Stock task. Because the choice task is iterative, there is no need to restrict the occurrences of either one of the tasks as would be done in a non-iterative choice task (such as the one depicted in Figure 4.11). Note that it is good practice to expand the Sell Stock task in the same manner as the Buy Stock task. However, because this is not particularly interesting for the ongoing discussion and to prevent overloading the figures, the Sell Stock task is kept concrete.

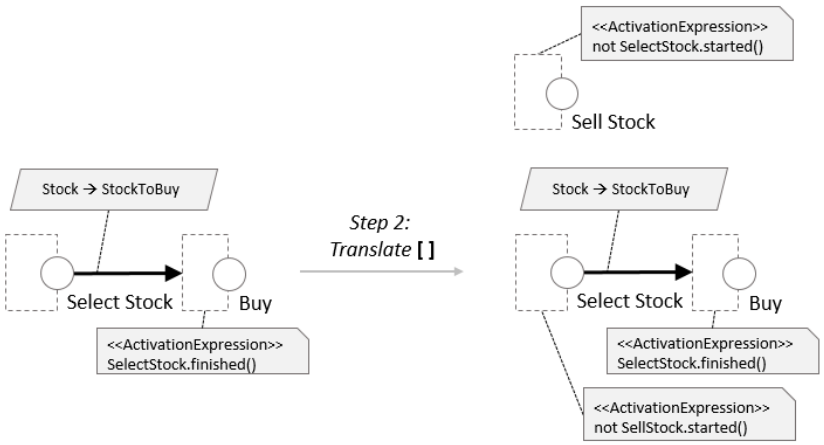


Figure 4.21: Step 2: Choice - Enabling translation combination

In Figure 4.22, the last step integrates the choice sub-tree with the interleaving operator at the root of the CTT tree. Because the use of the

interleaving operator implies that all its tasks must be accessible at the same time, to perform them in an interleaving manner, the abstract tasks of this operator are enclosed in the event space of the interleaving operator. All events and components of the outer event space can be accessed from the inner event spaces. The nesting event space ensures that when the user is, for example, selecting stock to buy, he can still monitor the stock rates while the buying process proceeds. Hence, the tasks in the initial enabled task set of the choice operator {Sell Stock, Select Stock} can be navigated to from the surrounding interleaving tasks. Similarly, the user can navigate from the tasks in the final enabled task set of the choice operator {Sell Stock, Buy} to the surrounding interleaving tasks.

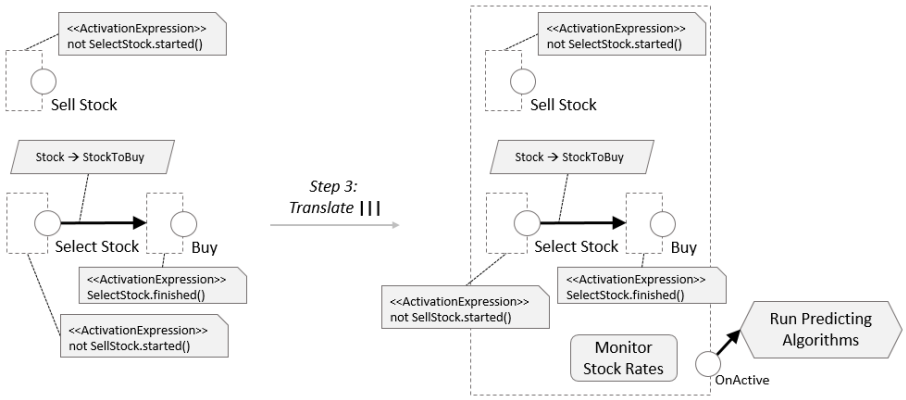


Figure 4.22: Step 3: Interleaving - Choice translation combination

A few of the many transformation combinations were elaborated upon. To be able to logically map the presented techniques which were used in the example combinations to other combinations, one more particular issue should be addressed. This issue affects all transformation patterns that accept application tasks. As seen before, an action, which is the translation of an application task, does usually not have more than two events (the NormalTermination and ExceptionalTermination events) attached. However, when a CTT tree as the one in Figure 4.23 needs to be translated, the action's NormalTermination event should reach both events spaces between which the user can choose. Because it is impossible to let the user decide which event space to go to upon the normal termination of an action, this termination event should navigate the user to an extra event space where the choice can be made by the user. This solution is presented in the same figure. Note that the extra event space, marked in red, only needs to be added in case an application task precedes a choice or order independent operator, since only those two have multiple entry points, as can be seen in light gray in

Figures 4.11 and 4.12. We will call the extra event space the *buffer event space*, as it forms a buffer which transforms a single incoming navigation flow into several outgoing flows.

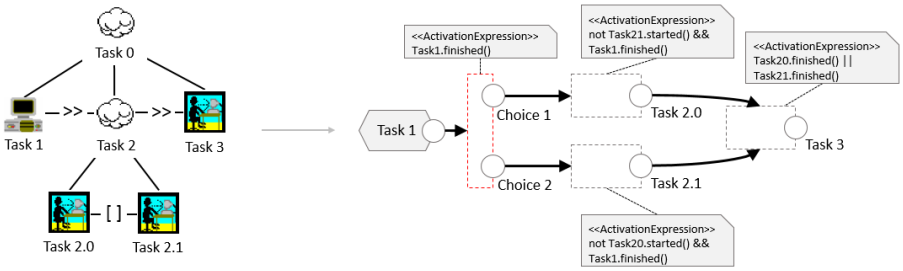


Figure 4.23: Application task input issue solved using a buffer event space

4.4.2 Translation of the ORM Object Chunks (Level 1)

So far, the CTT task trees for each of the informational and functional requirements of the system were translated into IFML. The Task & Information Modelling sub-phase covers the creation of both CTT trees and their accompanying ORM object chunks. The latter are not yet covered in the adapted Navigational Design sub-phase.

Object chunks are modelled per elementary CTT task. Most information they contain will be needed in the Implementation Design phase to, for example, specify which fields are needed in a specific input form in the user interface. Since those fields and forms in general do not belong to the Conceptual Design phase, the Object Chunks need to be forwarded to the Implementation Design phase. There they can be used to guide the interaction modelling at the implementation level, using standard IFML.

In the Conceptual Design phase, there is however one specific application left for the object chunks. Several CTT temporal operators support information exchange between their connected tasks (e.g., enabling with information exchange, interleaving with information exchange, and recursion). When this is the case, the translated conceptual IFML model contains parameter bindings or parameter binding groups attached to navigation or data flows between the corresponding IFML constructs. As the object chunks determine which data is the input and the output of an elementary CTT task, these input and output parameters are the only types of parameters that can be present in the mentioned IFML parameter bindings. Upon transformation of the CTT task trees, the object chunks hence need to be taken into

account to enter the correct parameters in the parameter bindings. To make a direct link with the object chunks, parameter names will always end with their type (e.g., **User** ->LoggedIn**User**). In Figure 4.24 an example is given in which a user logs in and is greeted by the system using his username in a greeting message. The object chunks for both the tasks are shown[6]. The output of the first task is the input to the next. As the object chunks are identified by their task's name, which is unique across the CTT model, they do not need to be visually linked to the IFML model and can be passed to the Implementation Design phase as an independent set.

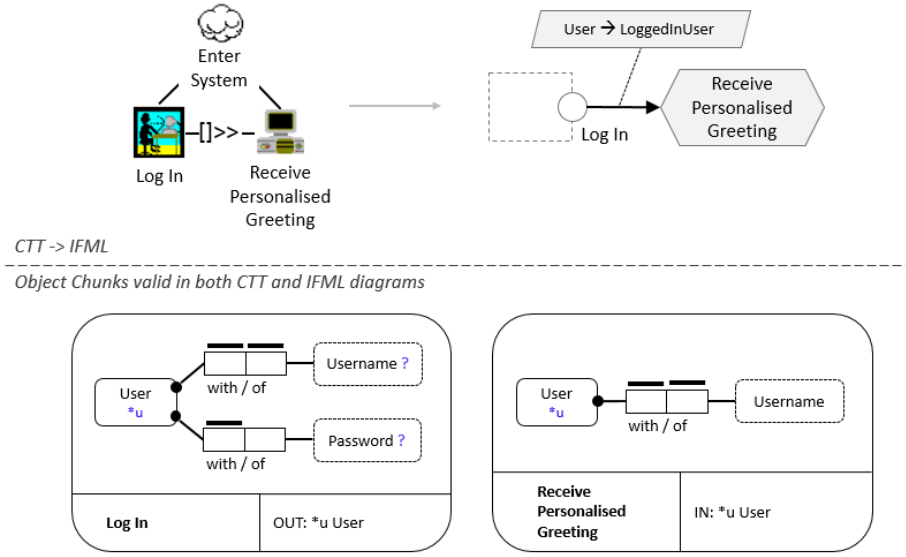


Figure 4.24: ORM chunks in the transformation process

4.4.3 Connecting the Obtained IFML Diagrams (Level 2 & 3)

The original navigational model specifies two higher levels: Level 2 and Level 3 (cfr. Figure 2.13). In Level 2, audience tracks are created using the models that each represent a single system requirement. Finally, Level 3 groups together the tracks and specifies the navigation between all of them. This level concludes the navigational modelling and contains the entire navigational

[6]The "?" operator in the Log In object chunk specifies that the lexical object type (here Username and Password) is for the user to input to the system. This operator is defined in De Troyer et al. (2008).

model. Both levels need to be reflected to the conceptual IFML approach as they are crucial in the system modelling.

De Troyer et al. (2008) specifies that when single-requirement models are composed into an audience track, if desired, it is possible to create a CTT task model which involves all abstract single-requirement tasks for the audience class. This is not always explicitly needed because there are systems in which the composition is straightforward and thus creating extra CTT trees would be overkill. However, to fully automate the transformations to conceptual IFML, which is a valuable property for the adjusted WSDM, using CTT models for this in every system design is favourable. This is due to the fact that, as shown earlier, all CTT trees have their corresponding IFML transformation. Hence, these same transformations can be used to automate the translation of the higher-level CTT models. As all means to translate CTT models are already in place and understood, imposing the developer to create higher-level CTT models does not make this WSDM phase more complex. On the contrary, it forces the developer to maintain structure within his web system design, which is a desirable property. To achieve this translation of an audience class's requirements as a whole, the following extra constraint is hence imposed in the Task Modelling sub-phase:

> *For each audience class, when task trees are created for all requirements of this class, create a task tree which contains these task trees and represents the complete system functionality available to that audience class.*

The task tree obtained after complying to this constraint will be called the *audience class tree*. An example which represents an application portal on a university website is shown in Figure 4.25. It comprises functionality for an Applicant audience class. Note that the leaf abstract tasks in the audience class tree refer to each of the audience class's requirement task trees. This extra tree is translated to IFML using the transformation patterns discussed earlier, taking the IFML results of the single-requirement task tree translations into account. The final IFML model will be called the *audience class IFML model*.

To this end, Level 2 of the original navigational model is now also covered. Before tackling the final level, Level 3, the audience class IFML models are tagged according to their audience class. As this needs to be visually modelled in the conceptual IFML diagram, to allow for later use in the Implementation Design phase, the audience class IFML models are presented in a grid of dashed lines which denote their boundaries. An example is depicted in Figure 4.26. Note that when the transformation from CTT trees

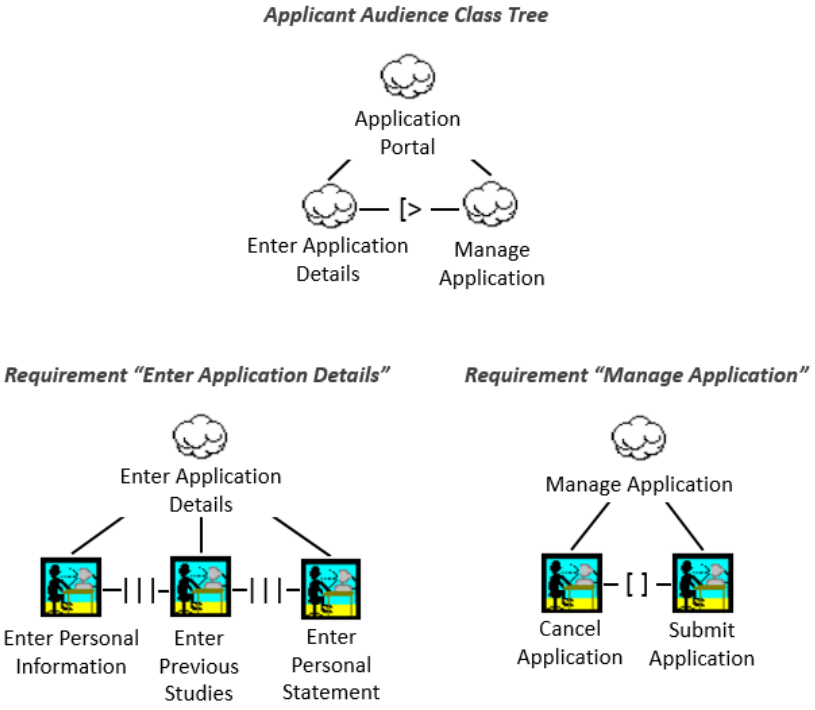


Figure 4.25: The audience task tree and the single-requirement task trees of the same audience class

to IFML diagrams is automated, the automated tool will need to draw these boundaries.

Level 3 of the navigational model connects the audience tracks according to the audience class hierarchy defined in the Audience Modelling phase. This implies that navigational facilities exist between each superclass-subclass pair in the audience class hierarchy. To keep the navigational modelling automated, once again the CTT notation can come to the rescue. The following constraint will be added to the Task Modelling sub-phase:

> *Upon construction of the audience class tree, of each audience class in the audience hierarchy, this tree must include abstract tasks which represent the audience class trees of its audience subclasses, if there are any, as specified by the audience hierarchy.*

This implies that, in the end, a single CTT tree represents the entire system under construction, called the *audience tree*. At the top of this tree, the Visitor audience class tree resides. This maps to the definition of the Visitor audience class, as every user who enters the system starts as a Visitor. All audience classes which are direct subclasses of Visitor are accessible

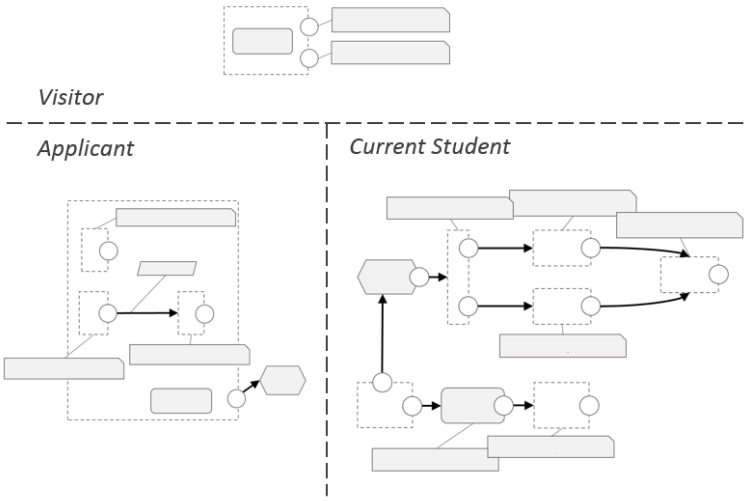


Figure 4.26: Audience class IFML models visually separated per audience class

from the facilities provided for visitors. This reasoning applies recursively to classes in a deeper level of the audience hierarchy tree. An example is presented in Figure 4.27. It shows the audience class trees of three audience classes in a university web system. The Visitor tree represents the top of the entire system tree, as explained before. At the bottom of the Visitor tree the two abstract tasks "Application Portal" and "Student Portal" represent the two other audience class trees in the figure, as denoted by the blue arrows. The other tasks in the Visitor tree are single-requirement tasks that were modelled separately and are not shown. Note that the tasks which represent logging in and out are not abstract. This is due to the fact that they do not require more detailed modelling and can be seen as single-requirement trees which only contain one task.

Level 3 also enhances the navigational model with navigational aid links which can be single links, a home link or landmark links to improve navigation possibilities within the web system. These links all need to be manually added to the IFML model by using IFML events and navigation flows, or by enhancing view components or event spaces with a "H" or "L" label. An example of these labels is shown in Figure 4.28. Note that these labels are not part of IFML, and that they are merely used to facilitate the transformations to standard IFML in the Implementation Design phase. They are hence temporary, just as the boundaries of the audience class IFML models are, as seen earlier. As the elements in the IFML models are not necessarily grouped into a single container which can be assigned a home link, multiple

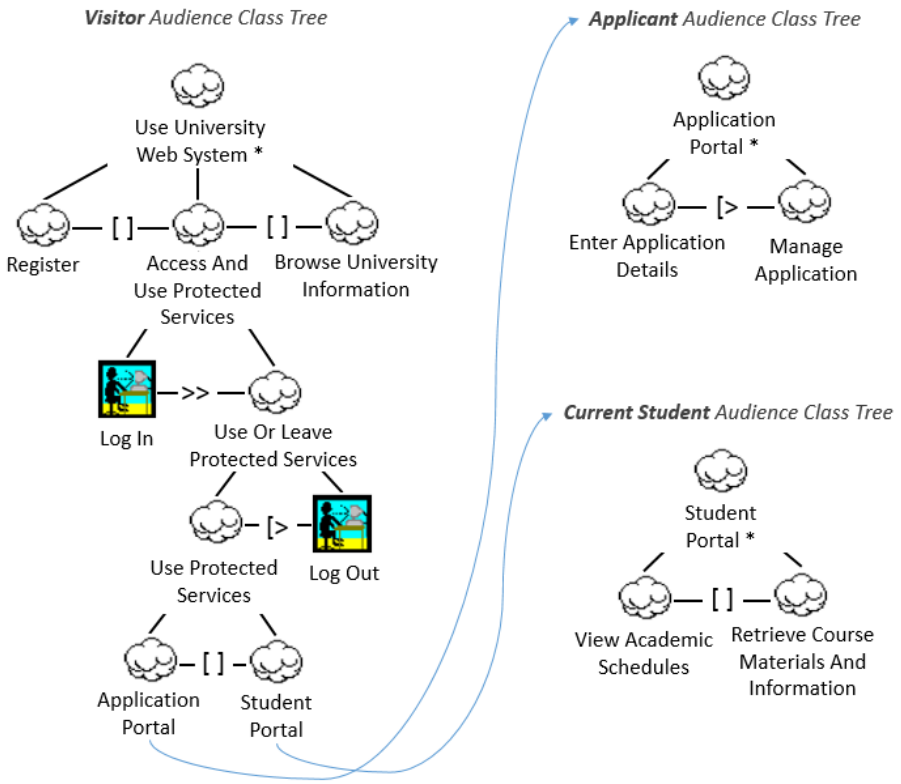


Figure 4.27: The audience tree construction

home links can be assigned to elements of the same IFML diagram.

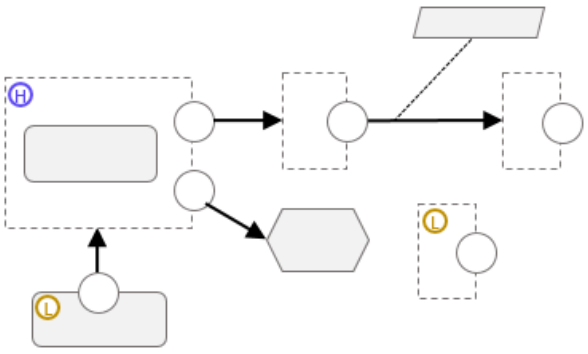


Figure 4.28: Navigational aid links and IFML

Note that WSDM also specifies a semantic link model which connects semantically-related object chunks. This model can be reflected in the final conceptual IFML model by adding extra navigation flows.

This concludes the integration of IFML in the Navigational Design sub-phase and in the Conceptual Design phase in general. The available outputs for the subsequent phases are the following: the navigational (conceptual) IFML model and the ORM object chunks.

4.5 IFML in the Implementation Design Phase

As the conceptual IFML is now described in detail, the discussion about where IFML fits into WSDM can be continued, taking into account that the Conceptual Design phase produces a navigational model in conceptual IFML. The next WSDM phase is the Implementation Design phase. It consists of three sub-phases: Site Structure Design, Presentation Design and Logical Data Design. In the following subsections it will be investigated to what extent the Interaction Flow Modelling Language can be used to replace or enhance these sub-phases. Note that the Logical Data Design sub-phase is not discussed as it relies on the modelled object chunks only and is therefore irrelevant for IFML integration.

4.5.1 IFML and the Site Structure Design Sub-phase

As discussed in the previous section, the adjusted Conceptual Design phase now makes extensive use of conceptual IFML. This results in a navigational model based on conceptual IFML. This IFML is not yet standard IFML as defined by the IFML specification (Object Management Group, 2015). For example, the conceptual IFML contains event spaces which were introduced to keep the IFML purely conceptual but are not part of the specification.

To be able to perform site structure design using the navigational conceptual IFML model, the conceptual IFML needs to be transformed into standard IFML. In fact, this transformation step yet includes the site structure design itself. By transforming the conceptual constructs in the IFML model into standard IFML constructs, the page structure of the web system (which is the main concern of the Site Structure Design sub-phase) is defined. This is due to the fact that IFML view containers are introduced into the model, and these specify which IFML components are grouped together on the same web page. During conceptual design, these constructs were avoided for this exact same reason, because determining the grouping of user interface elements in pages is implementation-specific. For example, for different device sizes (and consequently screen sizes), user interface elements can be grouped differently.

The transformation of the conceptual IFML model into standard IFML

does not only include structuring the web system into pages using view containers. For example, event spaces must also be transformed to more complex combinations of constructs to allow the task they support to be performed. The following adjustments need to be made to the conceptual IFML model in order to obtain a standard IFML model: concretisation of the conceptual event spaces, further pagination by introducing extra view containers, and translation of the navigational aid link tags to valid IFML constructs.

Next to these adjustments, the developer can also add extra details to the standard-IFML model in order to provide a more specific model to the Implementation phase. For example, adding a swipe event to a certain list view component (in a smart phone or tablet design) in order to browse the list by swiping the touch screen, or splitting a view component in two to support both browsing and searching a list. These details are enhancements which do not need to be specified in the web system's requirements explicitly, but are needed to provide the user an intuitive and usable interface.

All of the adjustments made to the conceptual IFML model constitute the entire transformation of this model into standard IFML and will be grouped in a single sub-phase, named the *Interaction Modelling* sub-phase. This sub-phase hence includes but is not limited to the site structure design and therefore replaces the Site Structure Design sub-phase. Each of the mentioned adjustments, made in the Interaction Modelling sub-phase, will be discussed in the following subsections.

Concretisation of Event Spaces

As mentioned earlier, in the conceptual IFML discussion, event spaces can be translated to 0 or more concrete IFML constructs. These constructs should support the purpose(s) of the event(s) attached to the event space. Due to the conceptual IFML transformation patterns, the attached events have either one or both of the following properties: they are used to navigate away from the event space, and/or they represent a task which can be performed in the event space. In case they are merely used for navigation, it is not necessary to attach any extra IFML constructs as the event itself will represent a control element in the user interface to perform the navigation (e.g., a button). The event space itself can then be replaced by a view container (Figure 4.29(a)). One particular situation in which this type of concretisation is useful, is when a buffer event space is used after an IFML action, as was seen in Figure 4.23. Since all choice events in the buffer event space are purely navigational, the buffer event space can be concretised to a view container.

In case an attached event supports an interaction task, extra constructs should be added for the task under consideration if appropriate. When no

extra construct is added to support the interaction task, the event space can be dropped (Figure 4.29(b)). If the extra constructs added are based on a single view component (possibly with extra attached events), this view component replaces the event space (Figure 4.29(c)). If on the contrary the interaction task requires more than one view component, the event space is translated into a view container which contains all of the needed view components and hence contains all facilities to perform the task (Figure 4.29(d)). Note that the last two event space translations (Figure 4.29(c) and (d)) translate the exact same event space in two different IFML models. This is due to the decision of the developer about whether he wants to represent the composition of a team by a table which can be edited in place, or a list which can be completed by filling in a form. This example marks the transition between conceptual and implementation-specific modelling.

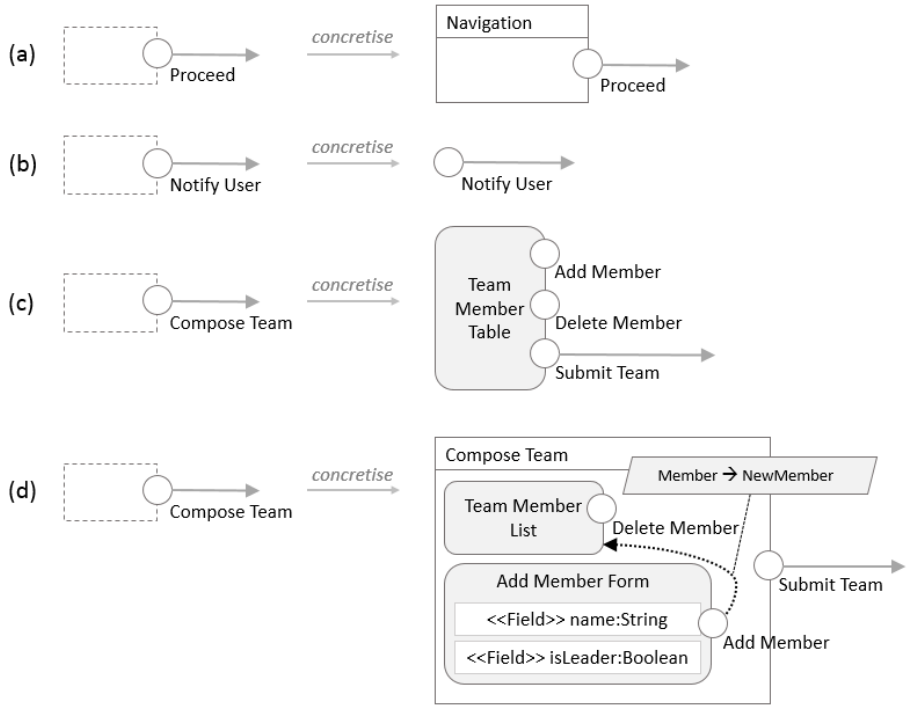


Figure 4.29: Event space concretisation examples

As mentioned earlier, when transforming conceptual IFML into standard IFML, the ORM object chunks should be considered. Figure 4.30 depicts an example. It concretises a small IFML diagram which was shown together with two object chunks in Figure 4.24. The diagram supports log-in functionality. As can be seen, the Log In ORM chunk is used to concretise the event space, as it indicates which parameters should be input by the user in the log-in

form. These result in the username and password view component parts in the concretised model.

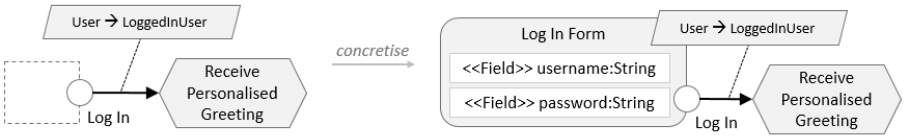


Figure 4.30: The use of object chunks in the concretisation process (based on Figure 4.24)

In some situations it might be natural to remove an event space rather than concretise it. To illustrate this, Figure 4.19, which shows deactivation and suspend-resume translation combination, can be reconsidered. The IFML model in the figure contains an event space to represent the deactivating task Close Model. This is useful if this task is complex (e.g., upon closing the model, the user should be asked whether he is really sure about closing it). On the other hand, the Close Model task could also just need a click on a button to be completed. In this case it does not make sense to keep the event space. Each of the suspend-resume tasks that can be deactivated already possesses a deactivation event. These can represent the close model button and hence the Close Model event together with its event space and activation expression can be dropped from the IFML model.

It is also possible to merge an event space or view component with a neighbouring one. In Figure 4.22, the last step of an CTT-to-IFML translation was presented. The two event spaces Select Stock and Buy are conceptually connected with a navigation flow. However, in the user interface buying stock might again be as simple as pushing a single button. If this is the case, both event spaces could be merged together so that selecting and buying stock happens in a single event space.

The fact that neighbouring event spaces need to be merged typically comes forth from the developer's CTT modelling style. It is not always straightforward how detailed a CTT model should be. In the example of Figure 4.22, the two merged event spaces were modelled as two separate CTT tasks. However, if the developer had modelled them in a single CTT task (e.g., "Select And Buy Stock") merging would not be necessary. Note that when removing or merging event spaces, the developer should be careful not to violate the system's requirements.

Not only neighbouring event spaces can be merged. A situation which often occurs on websites is that additional functionality is available to a logged-in user as opposed to a user which is not logged in. For example,

comments on YouTube can only be made by logged-in users. If such a situation is to be modelled, the main functionality will be modelled in the audience class IFML model of the visiting user, while the additional functionality resides in the IFML model of the registered user. Because both the main and the additional functionality should be close to each other in the user interface, both event spaces which comprise the total functionality can be merged. Hence merging event spaces across different audience class IFML models is possible too. Any set of event spaces or view components can be merged where appropriate.

In specific situations, event spaces contain other event spaces or view components (e.g., when interleaving abstract tasks such as in Figures 4.20-4.22). If this is the case, the developer should be extra careful in the concretisation process of the containing event space. Intuitively, the event space would have to be translated into a single view container. However, this container would specify the boundaries of a web page in which all the functionality inside the event space can be accessed. Often this might not be desirable, as too much functionality would be grouped onto a single page.

This problem can be solved in different ways. It is, for example, possible to concretise the event space into a XOR'ed view container and split up the content inside it into different view containers which are, due to the XOR constraint, never visible at the same time (Figure 4.31(a)). In the user interface, this might result in a tabbed view (on a single page). If this type of view is not appropriate to represent the functionality inside the event space, another possibility is to split up the content of the event space into several independent view containers, which are connected correctly to not break the functionality (Figure 4.31(b)). Note the two added navigation flows between the Task 1 and Task 2 view components in Figure 4.31(b). These allow navigation between the two interleaving interaction tasks. In this case the action Task 3 needs to be connected to both the separate view containers, to make sure it runs concurrently while performing both interaction tasks.

During event space concretisation it is allowed to rename any constructs in the model where appropriate. This might be needed due to the fact that constructs often have more active names such as "View Order Details" (caused by the translation from CTT notation), while in IFML the preferred notation for the view component needed to perform this viewing task would be "Order Details".

Additional View Container Pagination

After the event space concretisation step, the IFML model can be paginated further. This is done by adding extra view containers to it. The pagination is

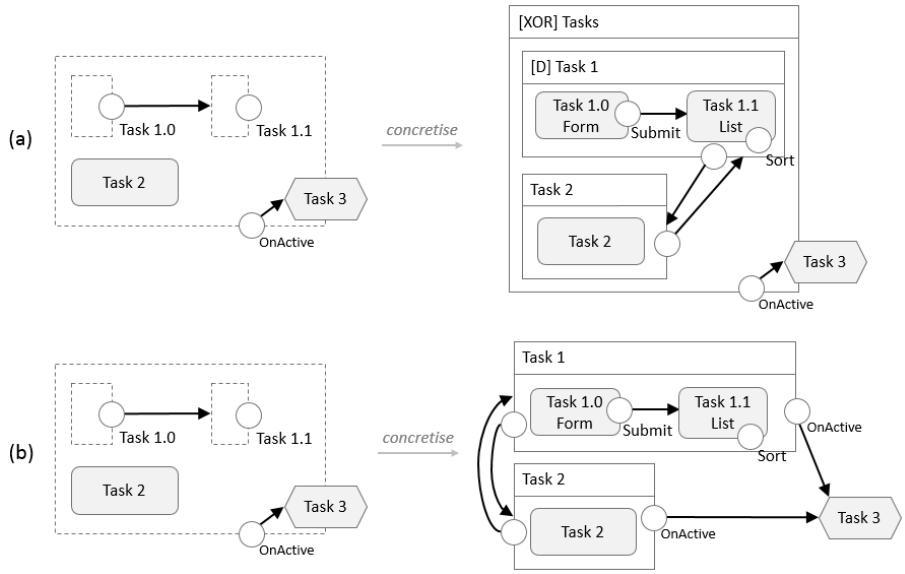


Figure 4.31: Concretisation of an event space which contains several other events spaces or view components

not complete until each IFML view component is assigned to a surrounding view container and thus belongs to a web page (view components are not meaningful outside of a view container). What should be mentioned is that when an entire system is modelled using interleaving tasks at the top of the audience tree, all view components will naturally reside in a view container yet due to the all-surrounding event space used in the interleaving transformation pattern. In this case, it might not be necessary to add additional view containers.

In this step, the developer can decide to group certain functionality on the same page. For example, if the developer thinks it is more appropriate to group two enabling tasks on a single web page rather than to spread them over two different pages in the mobile version of a website, he can do this in this step. This decision might enhance the usability of the mobile version of the system as the user only needs to request the page once rather than twice.

Until now, the access to IFML constructs that should not be enabled was prevented through extensive use of activation expressions. These expressions mainly guard the fact that different sets of IFML constructs cannot be enabled at the same time. For example, in a step-by-step task process (denoted as a series of view components connected pairwise by navigation flows), it is often desirable to consider a step as “complete” when the user navigates to the next step. The completed step should disappear out of the sight of

the user and the next step should be presented. The view container corresponding to the completed step should in this case be deactivated. Until now, this deactivation was covered by the activation expressions. However, as view containers are now about to be added to the model, many of these activation expressions are no longer useful. This is due to the fact that view containers group elements on a single page and thus already constrain that elements in different view containers cannot be active at the same time[7]. To illustrate this, consider the example in Figure 4.32. Both the concretisation and pagination steps are performed on the presented conceptual model. As can be seen, the activation expression marked in red disappears in the last step. As the stock buyer's form is now on a different page than the stock selection list, the activation expression is no longer meaningful.

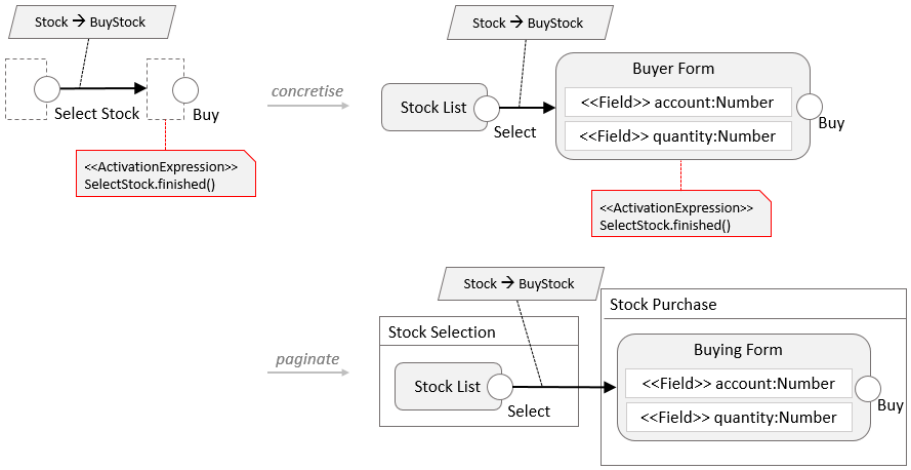


Figure 4.32: Concretisation and pagination example: activation expression disappears as view containers are introduced

In the previous WSDM phase, certain activation expressions were created which contain code such as `Task.started()`, `Task.ongoing()` or `Task.finished()`. The `Task` object of these operations is a remainder of the CTT notation, as the developer thinks in terms of tasks. In IFML, these operations do not always make sense as the language does not have a "task" concept. Instead, a task can comprise one or more IFML user interface and interaction constructs. Until now it was useful to express these operations in terms of tasks because each elementary task existed solely of a single conceptual IFML construct. However, now these operations should be adjusted in order to fit into the standard IFML model.

[7]As the user can only see one page at a time.

Often these adjustments are simple. For example, `Task.started()` and `Task.ongoing()` can mostly be expressed as the disjunction of the `hasFocus()` function over the set of IFML constructs provided to complete `Task` (i.e., `C1.hasFocus() || C2.hasFocus() || C3.hasFocus()`, supposing that C1, C2 and C3 are the three constructs which aid in completing the task). The `hasFocus()` function can be understood as a conditional function which returns true if the object it is called upon is currently accessible to the user in the user interface, and false if not. Thus, when any of the constructs to complete the task is accessible, the user could be working on the task.

In the case of the `finished()` operation, the developer should determine when the task is logically finished. For example, when a submit button is clicked after filling in some form, the task can be considered as finished. In this example, the `Task.finished()` condition could then be replaced by `Form.submitted()`.

We choose to perform these activation expression adjustments after the pagination of the model, as pagination eliminates a significant amount of activation expressions. It does not make sense to adjust the expressions to then delete them right after.

As a guideline, we like to mention that it is important that the developer is well acquainted with the system's requirements during the pagination step, as mistakes against these requirements are easily made when modifying the IFML model without care.

Navigational Aid Link Tags Translation

Two types of navigational aid link tags were used to denote view component or event space access from anywhere within the system: the home link and the landmark links. They can be translated to the view container tags "[H]" and "[L]" respectively. Note that the home tag H is not included in the standard IFML, but could be created through an extension, as described in Section 2.1.4. It could be an extension of the landmark view container of which there must be exactly one instance and which is regarded as the landing page of the web system.

To translate the navigational aid link tags, the developer should assign the appropriate view container tag to the most outer view container in which the element which possesses the tag resides. After concretisation and pagination of the model, all view containers will be available. Therefore, this is the appropriate time to translate the tags. Note that the most outer view container receives the tag because it typically corresponds to the web page, whereas the more inner ones represent parts of the web page.

This covers the case in which the tagged view component or event space

resides in a single view container. If an event space would be spread over different containers, it is advised to pick a single container to assign the tag to. This container must possess the user interface elements which are needed at the start of the tasks to be completed in the event space. If no such elements exist, any container can receive the tag.

4.5.2 IFML and the Presentation Design Sub-phase

In the Presentation Design sub-phase, page templates and models are created to specify the exact look and feel of each page in the web system. IFML cannot be used directly to support the creation of these models. However, the use of IFML in the previous phases has a noteworthy impact on the Presentation Design sub-phase: the page models are partially created yet. Indeed, the IFML diagram already covers which types of user interface elements will be present on all of the web system's pages. All what is left to do is to give these elements a concrete look and feel, and structure them visually using a grid, as in the original WSDM sub-phase.

Due to the boundaries drawn between the audience class IFML models in the Navigational Design sub-phase, it is possible to discriminate between the audiences who have access to the pages in the final IFML model. This helps the developer in designing page templates, as he can now take the usability requirements and user characteristics of the audience class into account.

4.6 Towards an Implementation

All WSDM phases have now been investigated and integrated with IFML where possible. Following the adjusted WSDM, the web system can now be implemented in the same fashion as in the original WSDM. In fact, it is possible to use the WebRatio tool discussed in Section 3.1 to generate the implementation entirely. The final IFML model obtained as output of the Interaction Modelling sub-phase could be directly exported to WebRatio. After transforming the presentation designs into WebRatio-compatible formats, linking the IFML model with the appropriate data sources and possibly adding implementation-level WebRatio-specific components to the model, the web-system can be generated in one click.

4.7 Overview of the Adapted WSDM

In this last section of Chapter 4, we present a scheme that summarises the procedural steps to be taken in each of the phases of the adapted WSDM,

which were discussed in detail in this chapter:

- Mission Statement Specification (as in original WSDM)
- Audience Modelling (as in original WSDM)
- Conceptual Design:
  - Task Modelling:
    - Create task trees per system requirement
    - Use these single-requirement trees to construct an *audience class tree* per audience class (taking into account that audience super-classes have their audience sub-classes' task tree as abstract tasks in their own task trees)
    - All audience class trees now form the *audience tree* which specifies all tasks to be performed in the entire web system
  - Information Modelling (as in original WSDM)
  - Navigational Design:
    - Apply the *task tree translation algorithm* to each single-requirement task tree (making use of the *transformation patterns*) to obtain conceptual IFML diagrams for each system requirement
    - Apply the same algorithm to the audience class trees, using the single-requirement IFML diagrams, to obtain an *audience class IFML model* per audience class (taking into account that audience super-class IFML models include their audience sub-classes' IFML models)
    - Overlay the audience class IFML models by a grid to visually separate them one from another (grouping the functionality added only by the respective audience class, audience subclass functionality is excluded)
    - Add navigational aid links and tags to the conceptual IFML model
    - Create a semantic link model and reflect it on the conceptual IFML model by adding extra flows
- Implementation Design:
  - Interaction Modelling:

        * Remove or merge event spaces and view components (optional)
        * Concretise the event spaces (taking into account the object chunks) and rename constructs where appropriate
        * Paginate the IFML model more by adding additional view containers (optional)
        * Revise the activation expressions in the model and adjust them where needed
        * Translate the navigational aid link tags by assigning a corresponding view container tag to the most outer view container in which the element with the tag resides
        * Add extra details to the IFML model (optional)
    - Presentation Design: as in original WSDM, only less work to be done due to the use of IFML
    - Logical Data Design (as in original WSDM)
- Implementation (as in original WSDM)

Using this scheme, the developer should be able to complete the IFML-based WSDM process successfully. It can hence be used as a guideline while developing. At last, we present Figure 4.33 which depicts the changes made to the original WSDM process presented in Figure 2.9. The yellow colour of a sub-phase denotes that the sub-phase was adjusted. The green colour denotes a new sub-phase. The blue dashed rectangle marks in which phases IFML was integrated.

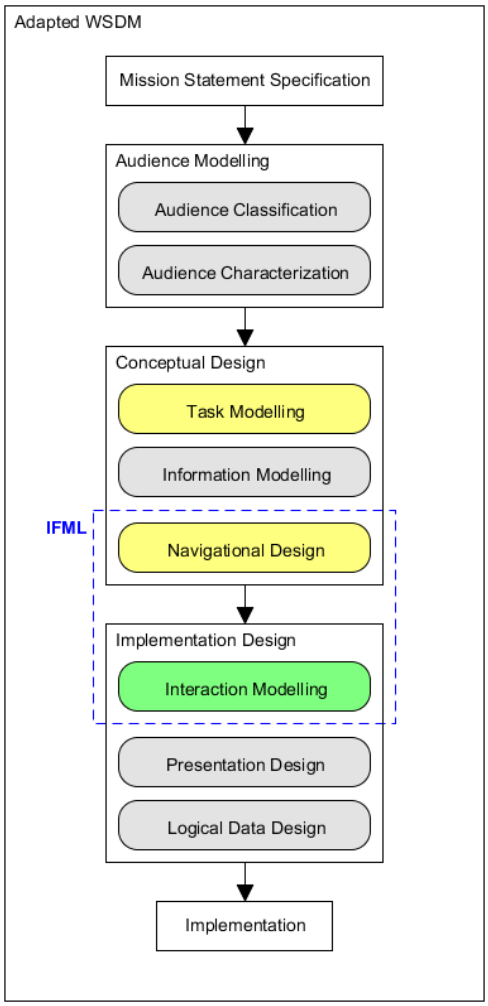


Figure 4.33: Structure of the adapted WSDM

5
Case Study

In the previous chapter, an adapted WSDM was proposed which integrates IFML into several of its phases. Concrete partial examples of IFML transformations and concretisations were given throughout this discussion. Now, to give the reader insight in how a complete adapted-WSDM design process is conducted, a case study is presented which spans each of the adapted phases.

For the sake of comparison, the case study is based on the Internet Movie Database[1] (IMDb) example used in De Troyer et al. (2008), which describes the original Web Semantics Design Method. In this paper, the website is extended with a games section, however this extension will not be considered in this case study.

In the following sections of this chapter, an IMDb-like web system that consists of a reduced set of the functionality of the original IMDb system will be modelled. To focus on the adapted parts of WSDM, the case study starts at the Conceptual Design phase, as it is assumed that the audience hierarchy and a set of requirements are constructed already. If more details on the first two WSDM phases are required, we refer the reader to De Troyer et al. (2008). The case study ends at the Presentation Design sub-phase.

[1] http://www.imdb.com/

5.1 Case Study Input

This section presents the audience hierarchy and the requirements of the audiences which together serve as the input of the case study. The content of this section is largely based on De Troyer et al. (2008).

Audience Hierarchy

There are two audience classes, the Visitor and the Movie Enthusiast, as can be seen in Figure 5.1.

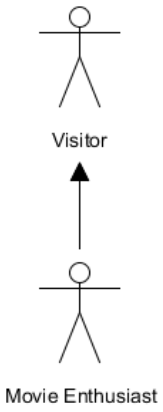


Figure 5.1: Audience hierarchy of the IMDb case study

Requirements

For both audience classes in the audience hierarchy the functional and informational requirements are presented in the following list.

- Visitor: a user from this audience class should be able to...
  - register for a movie enthusiast account (1)
  - log in to a movie enthusiast account (2)
  - view the movies currently playing in the cinemas (3)
  - buy tickets for movies that are currently in the cinemas (4)
  - search for information about a specific movie (5)
- Movie Enthusiast:

– log out from the movie enthusiast account (6)
– add a movie to his to-watch list (7)
– manage his to-watch list (8)

5.2 Adapted WSDM: Conceptual Design Phase

In the previous section the case study input was created according to the original WSDM process. The constructed models are now used as input to the first adapted WSDM phase: the Conceptual Design phase. In the following subsections, each sub-phase of the Conceptual Design phase is treated.

5.2.1 Task Modelling

For each of the requirements in the previous section a CTT tree is created. Some requirements, which are closely related, might be grouped together in a single CTT tree (e.g., requirements (3) and (4)). Figures 5.2 through 5.5 depict the task trees for the given requirements (the first two figures of this series were based on De Troyer et al. (2008)). Note that the following requirements, which can be captured in a single task, are not shown: (1), (2) and (6). Each of these requirements can be represented by a single state-changing interaction task.

To complete the task modelling, the single requirement task trees are used to construct audience class trees for both the Movie Enthusiast and Visitor audience classes, depicted respectively in Figure 5.6 and 5.7. The two resulting trees together form the audience tree.

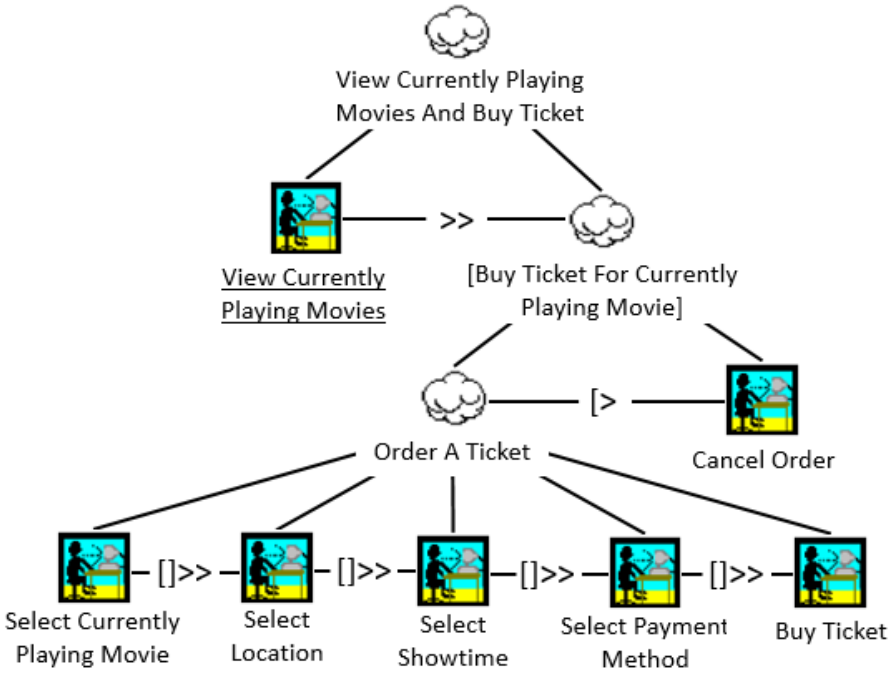


Figure 5.2: CTT task tree for requirements (3) and (4)

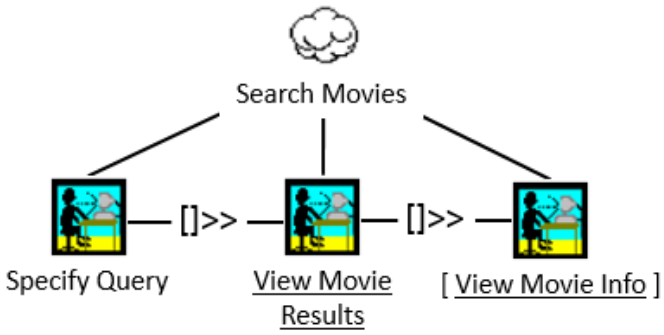


Figure 5.3: CTT task tree for requirement (5)

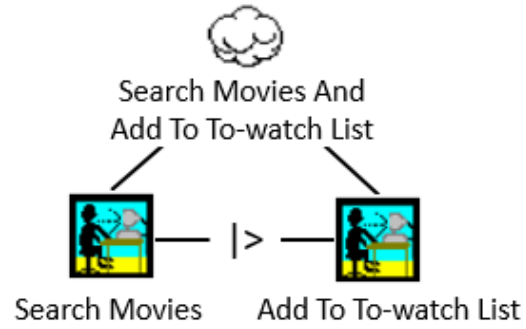


Figure 5.4: CTT task tree for requirement (7)

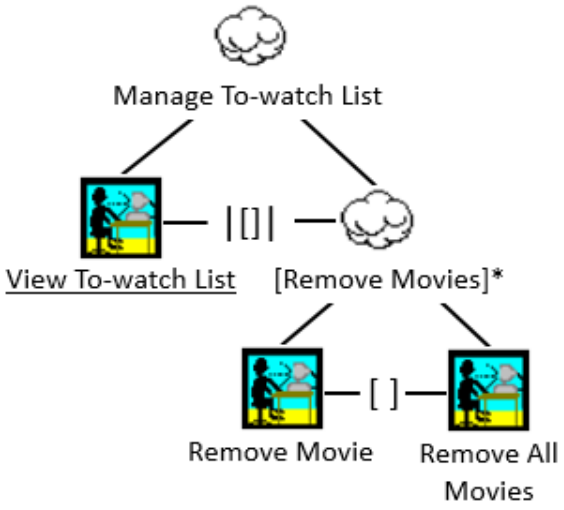


Figure 5.5: CTT task tree for requirement (8)

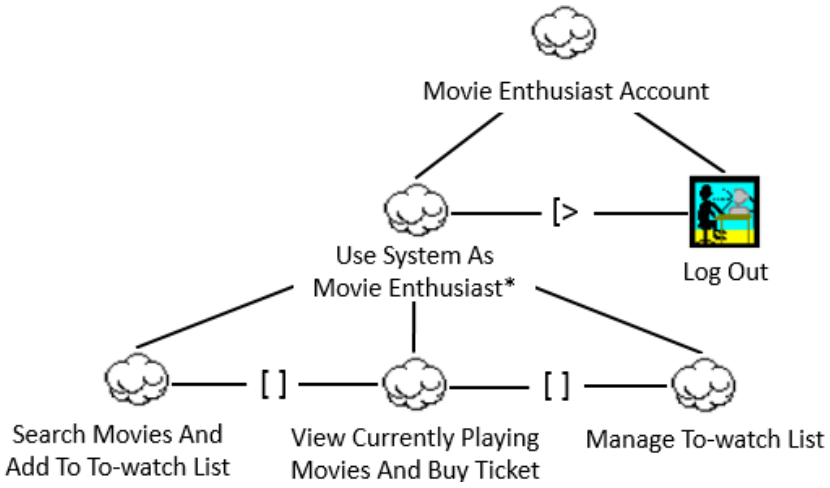


Figure 5.6: Movie Enthusiast audience class tree

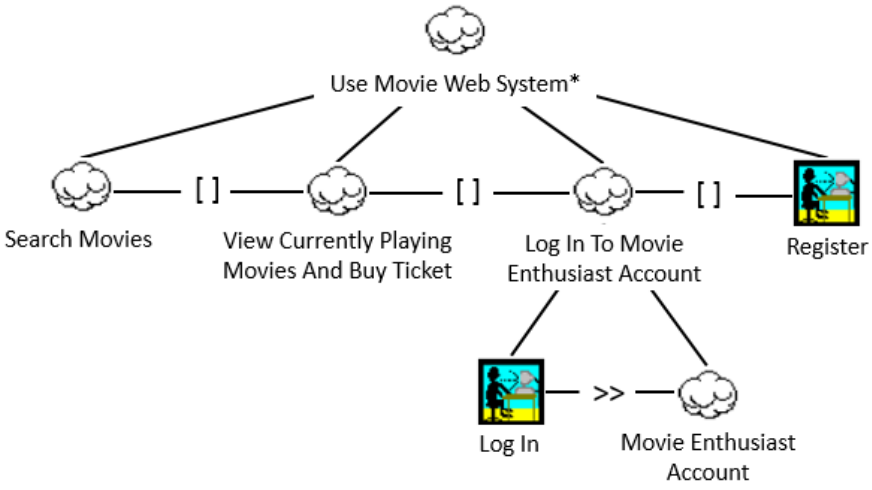


Figure 5.7: Visitor audience class tree

Object Chunks

The object chunks which accompany the elementary tasks "Select Location" and "Select Showtime" in the CTT task tree for requirements (3) and (4) (Figure 5.2) are presented in Figures 5.8 and 5.9. To not overload this section, only the object chunks for these two elementary tasks are shown.

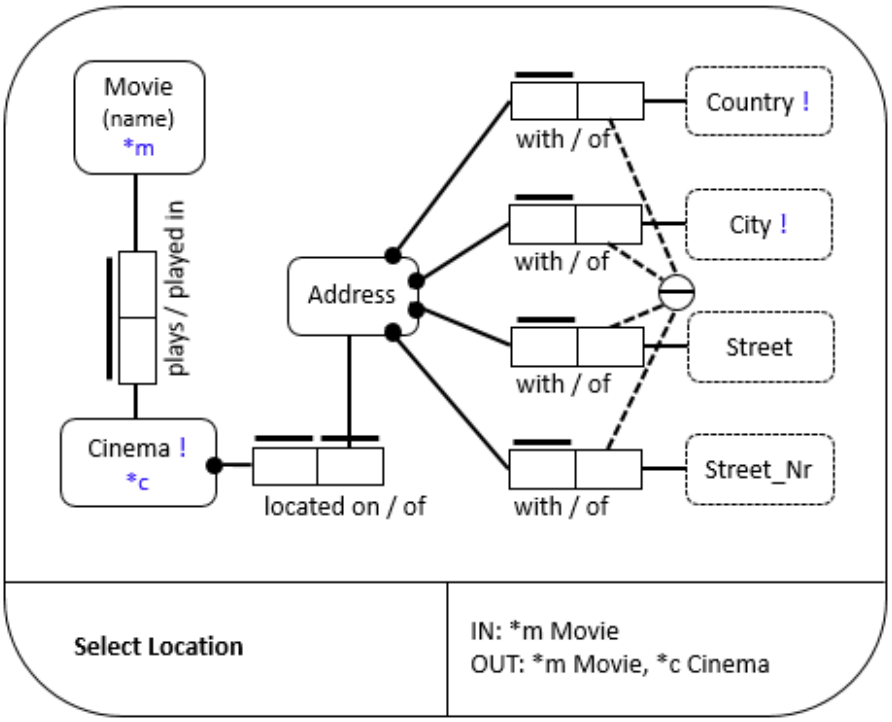


Figure 5.8: ORM object chunk for task "Select Location"

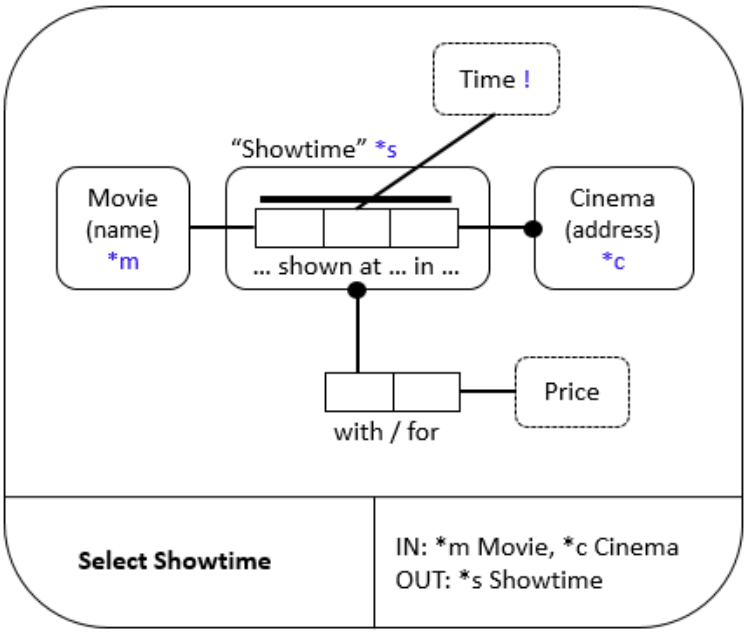


Figure 5.9: ORM object chunk for task "Select Showtime"

5.2.2 Navigational Design

The next step in the Conceptual Design phase is the Navigational Design. First, all single-requirement trees need to be transformed into conceptual IFML diagrams. Figures 5.10 through 5.13 depict the results of this transformation.

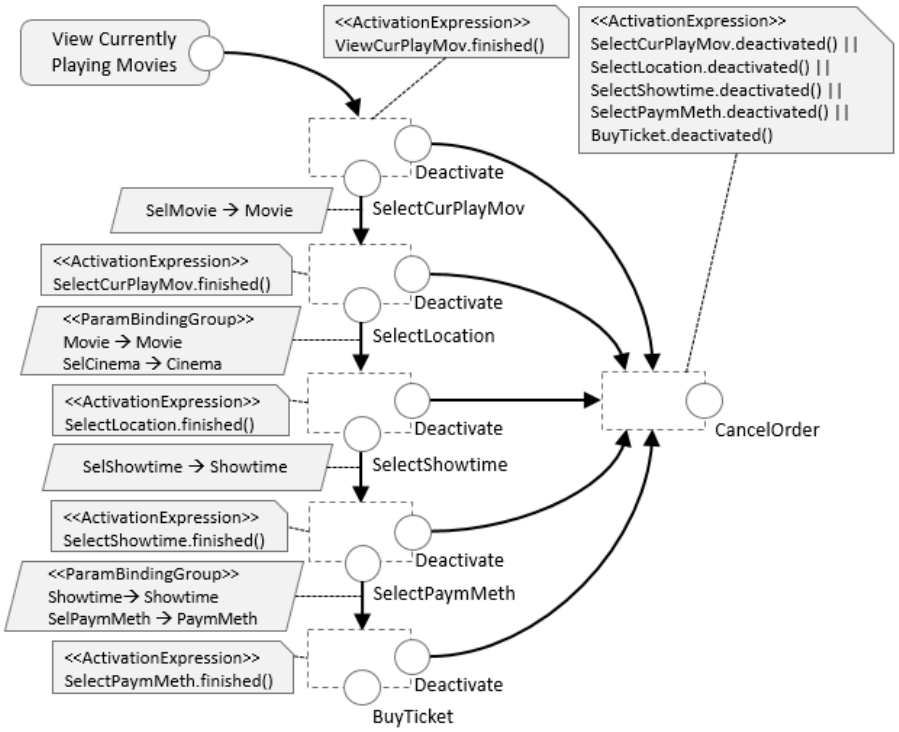


Figure 5.10: IFML diagram for CTT tree "View Currently Playing Movies And Buy Ticket"

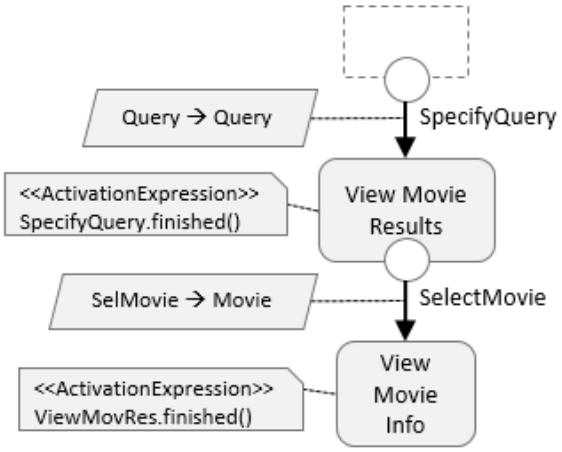


Figure 5.11: IFML diagram for CTT tree "Search Movies"

In Figure 5.12 the "Search Movies" transformation is identical to the transformation in Figure 5.11, which is due to the fact that the task is reused in two different requirement task trees. When composing all requirement IFML models, which is done in the next step of the navigational design, there will be no duplicate IFML models due to this reuse, as both requirement models will be mapped onto the same "Search Movies" IFML piece which will reside in the Visitor audience IFML model.

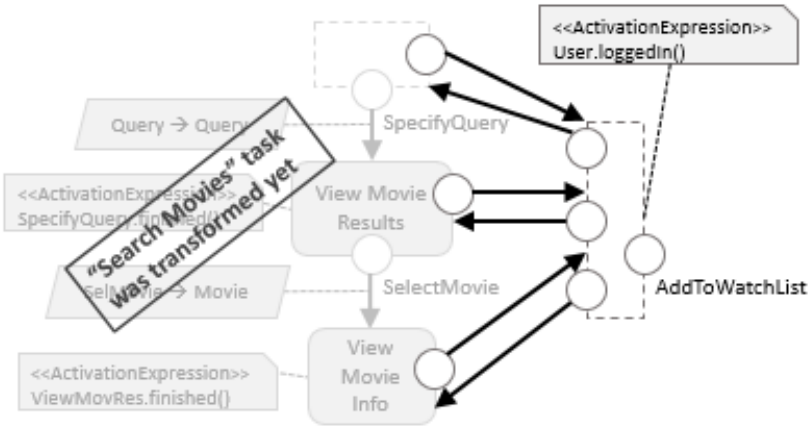


Figure 5.12: IFML diagram for CTT tree "Search Movies And Add To To-watch List"

All requirement IFML models are now in place. The next step is to transform the audience class trees into audience class IFML models using the obtained requirement models. Figures 5.14 and 5.15 depict the audience

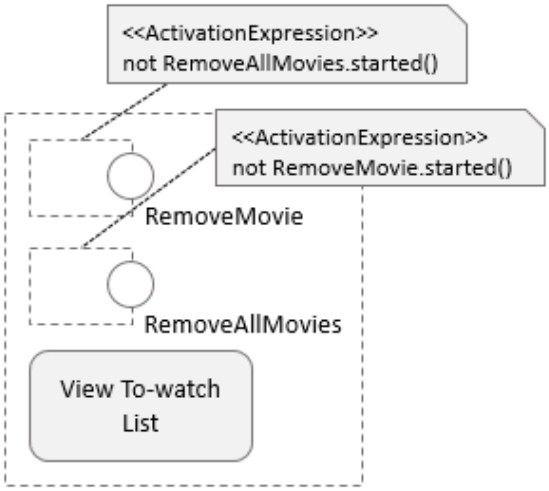


Figure 5.13: IFML diagram for CTT tree "Manage To-watch List"

class IFML models for the Movie Enthusiast and Visitor audience classes respectively. Note that the Visitor model does not contain the Movie Enthusiast model yet. Both will be merged in the next step. Neither does the Movie Enthusiast model contain all parts that are also modelled in the Visitor model (e.g., "View Currently Playing Movies And Buy Ticket"). This is to prevent unnecessary duplication, as in the merge step the entire model will be shown.

It can also be noticed that a landmark tag appears on the "Log Out" event space in Figure 5.14. As this event space must be available from all other event spaces and view components in the model, it is already assigned a landmark tag. This prevents adding and subsequently removing a large set of navigation flows.

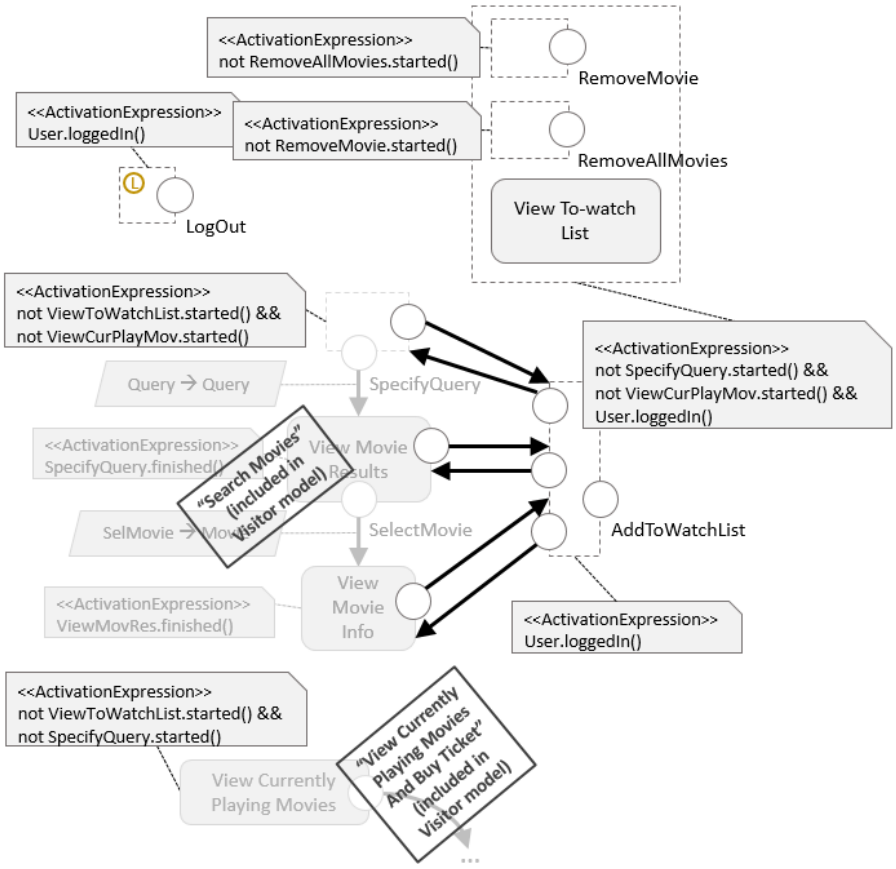


Figure 5.14: Movie Enthusiast audience class IFML model

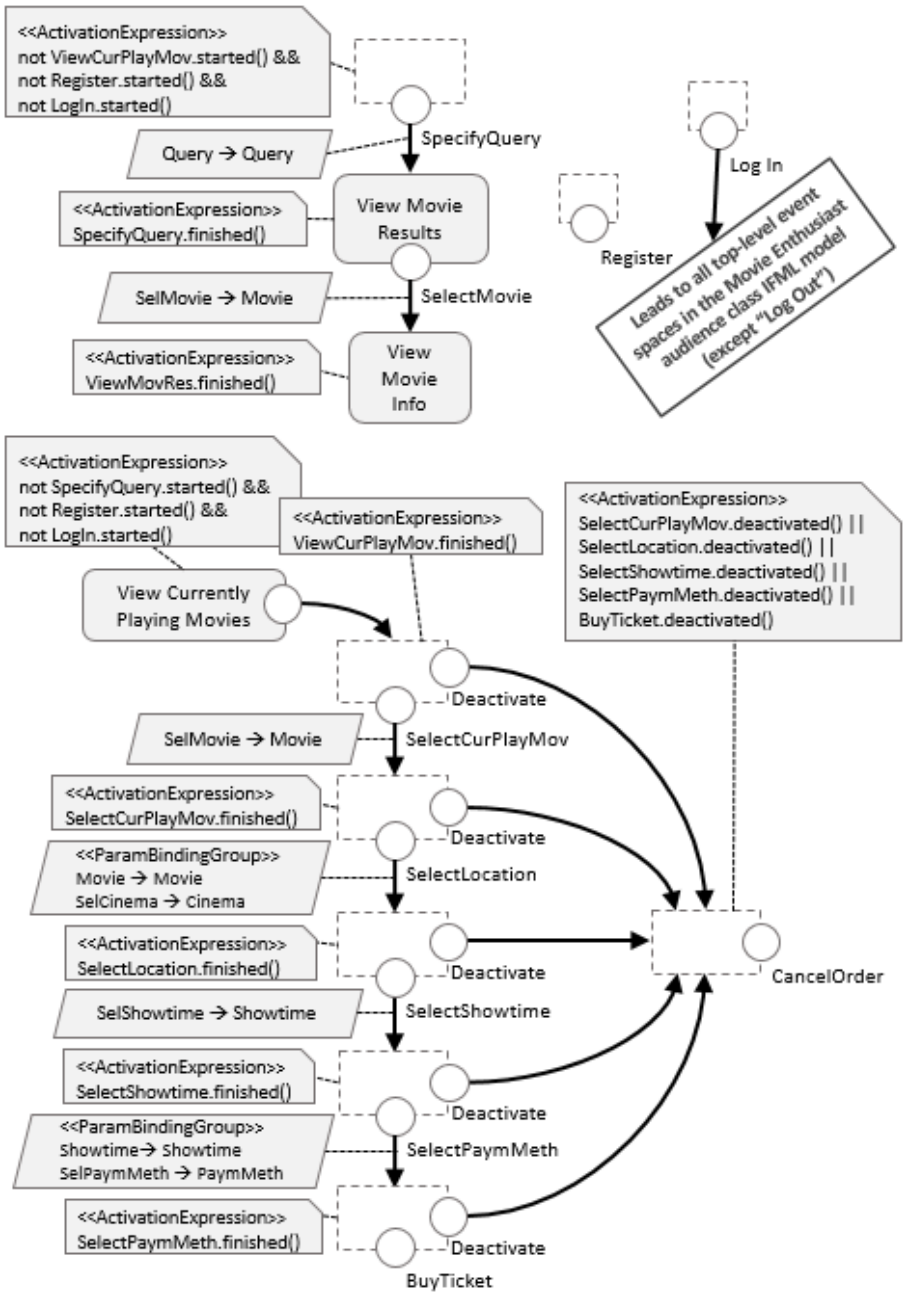


Figure 5.15: Visitor audience class IFML model (Movie Enthusiast part left out)

Figure 5.16 depicts the complete audience IFML model after merging the Movie Enthusiast and Visitor audience class IFML models. A grid is drawn to visually separate the models. Also navigational aid tags and links are added to the model. This model concludes the Navigational Design sub-phase and thereby the Conceptual Design phase. The outputs of this phase are the model given in Figure 5.16 and the object chunks.

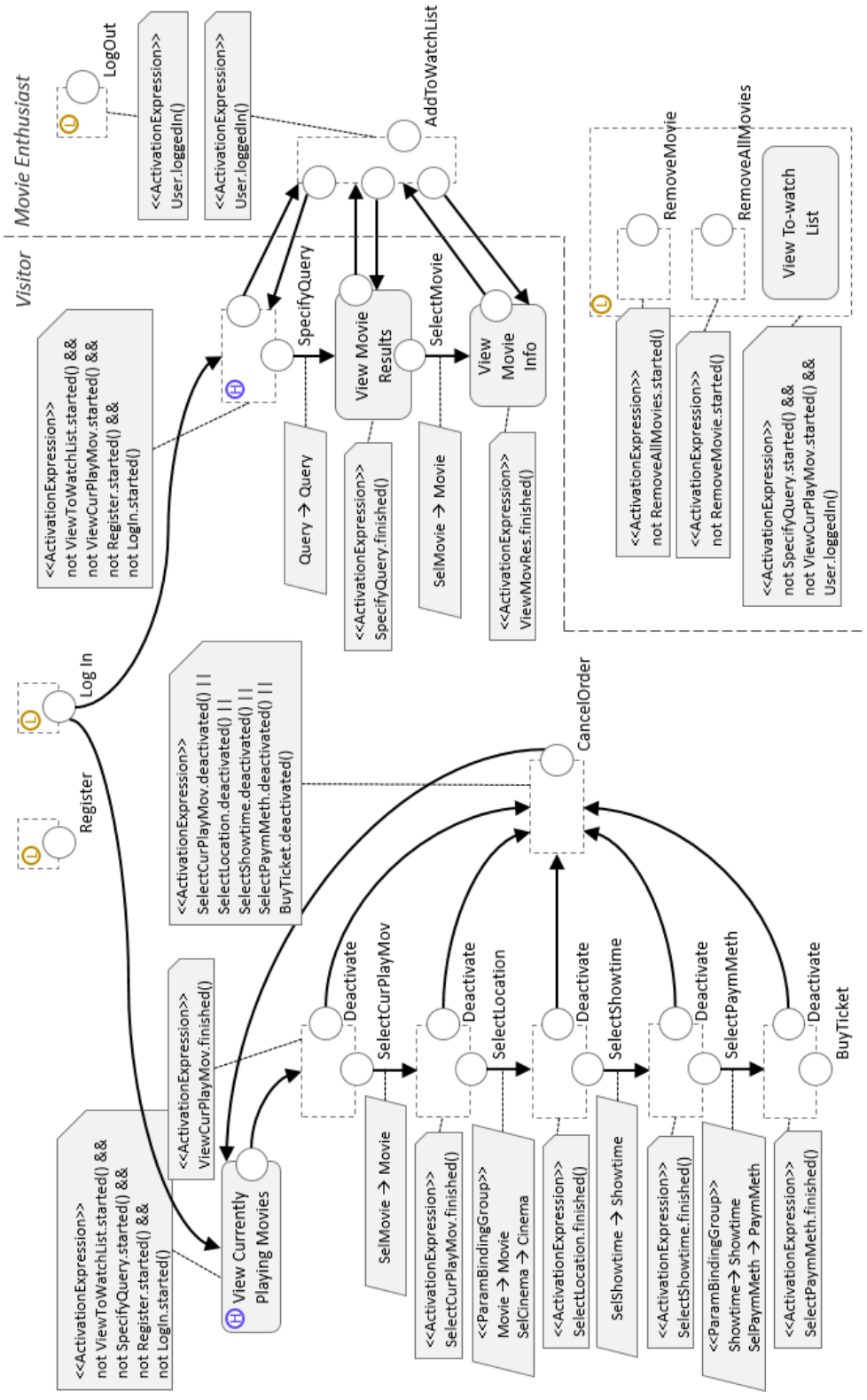


Figure 5.16: The audience IFML model

5.3 Adapted WSDM: Implementation Design Phase

The next phase is the adapted Implementation Design phase. In this section, the new Interaction Modelling sub-phase will be illustrated. Finally, the homepage of the web system will be shown in the Presentation Design sub-phase section to provide insight in how the final system might look.

5.3.1 Interaction Modelling

The first step to be taken during interaction modelling is removing or merging event spaces and view components in the audience IFML model. This step is optional, however useful for this case study. Figure 5.17 shows the resulting model after performing these operations. As can be seen, the movie removal event spaces are removed and their corresponding events are attached to the "View To-watch List" view component. Also the "Cancel Order" and "Add To To-watch List" event spaces are removed and merged with the event spaces which they respectively deactivate or suspend.

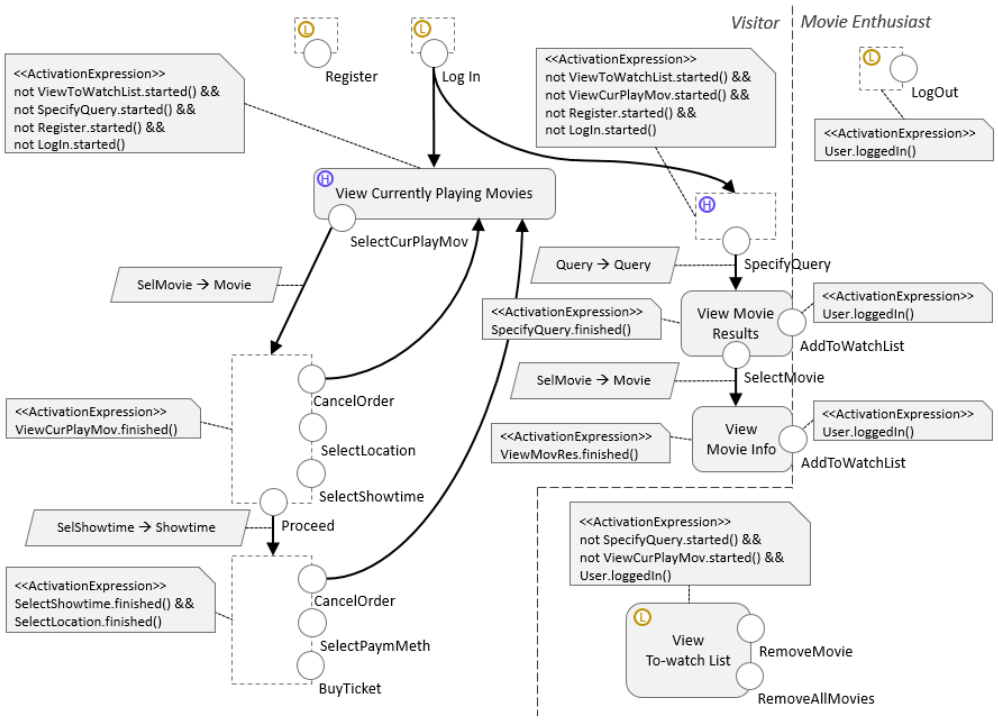


Figure 5.17: Event spaces removed

The next step is to concretise the event space in the model. Figure 5.18 shows the result after this step. Note that also the naming of some of the constructs is adjusted to conform more closely to IFML notation.

The model now looks a lot more like standard IFML yet. Only the page structure is not complete, and the navigational aid tags still need to be translated. This is what is done next. Figure 5.19 shows the final result of the Interaction Modelling sub-phase after carrying out these operations: a correct IFML diagram.

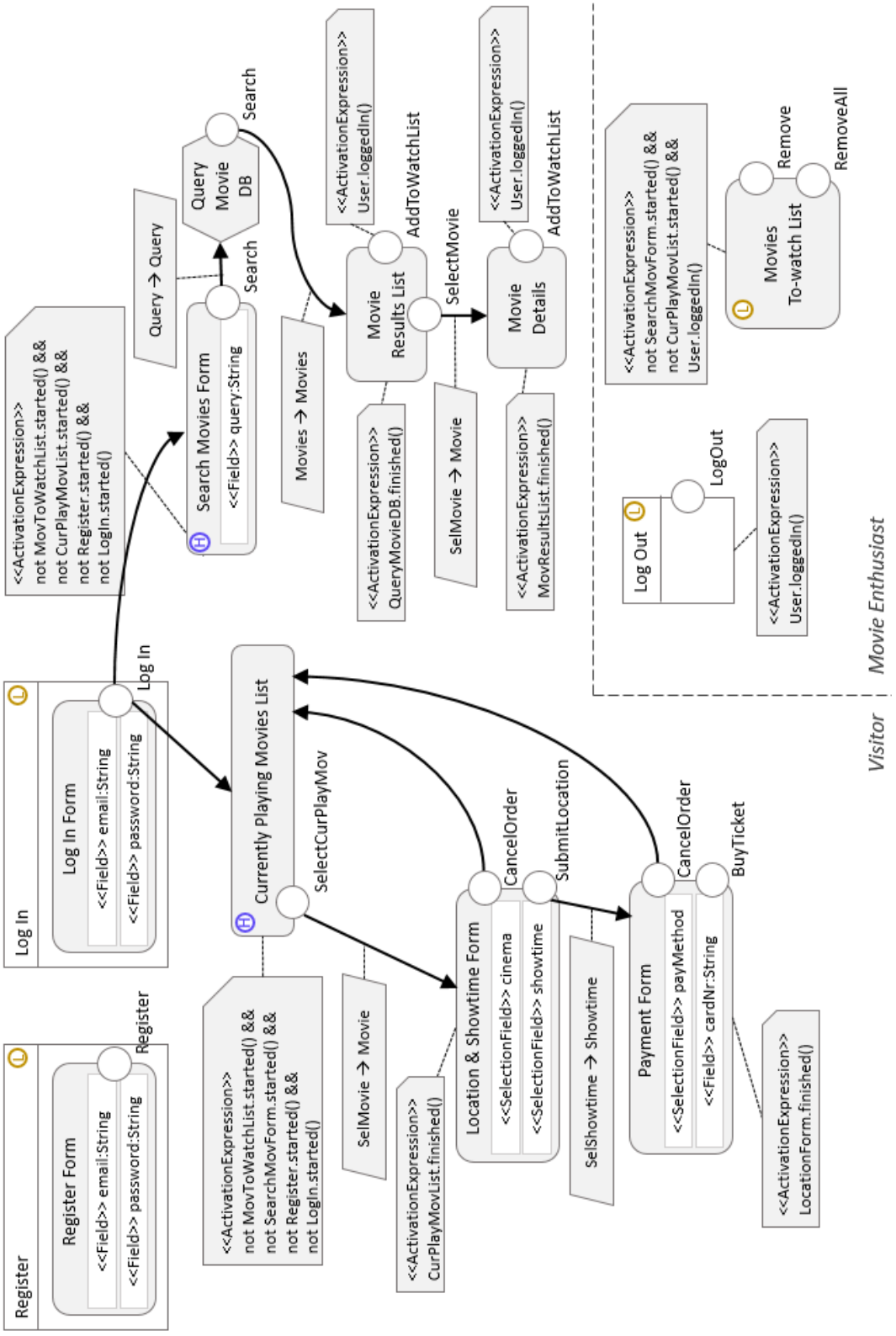


Figure 5.18: After event space concretisation and renaming

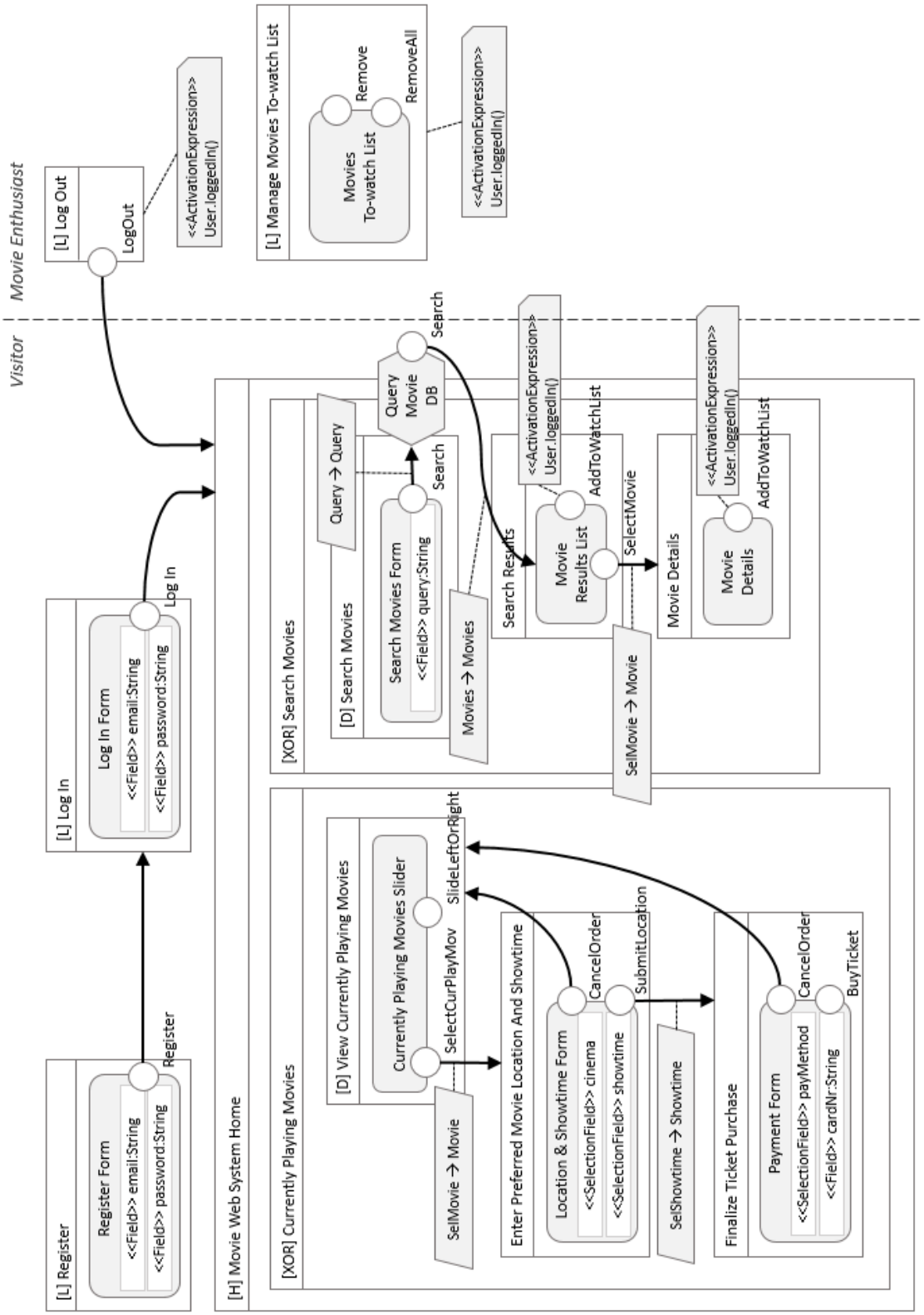


Figure 5.19: The final standard IFML model

5.3.2 Presentation Design

As final part of this case study, a page model for the Visitor's homepage of the movie web system is presented in Figure 5.20. This model is based on the IFML diagram created in the previous sub-phase of the Implementation Design phase. The two XOR view containers in the home page container, which represent the movie search and the currently-playing movie browse facilities, are presented in the user interface one above the other. The currently-playing movies can be browsed by means of a slider, as was specified in the last step of the previous sub-phase. To start the ticket buying process, the user can click the "Buy Ticket" button that accompanies the movie that is currently in the slider's focus. The landmarks which are available to the Visitor are shown in the menu bar on top of the page.

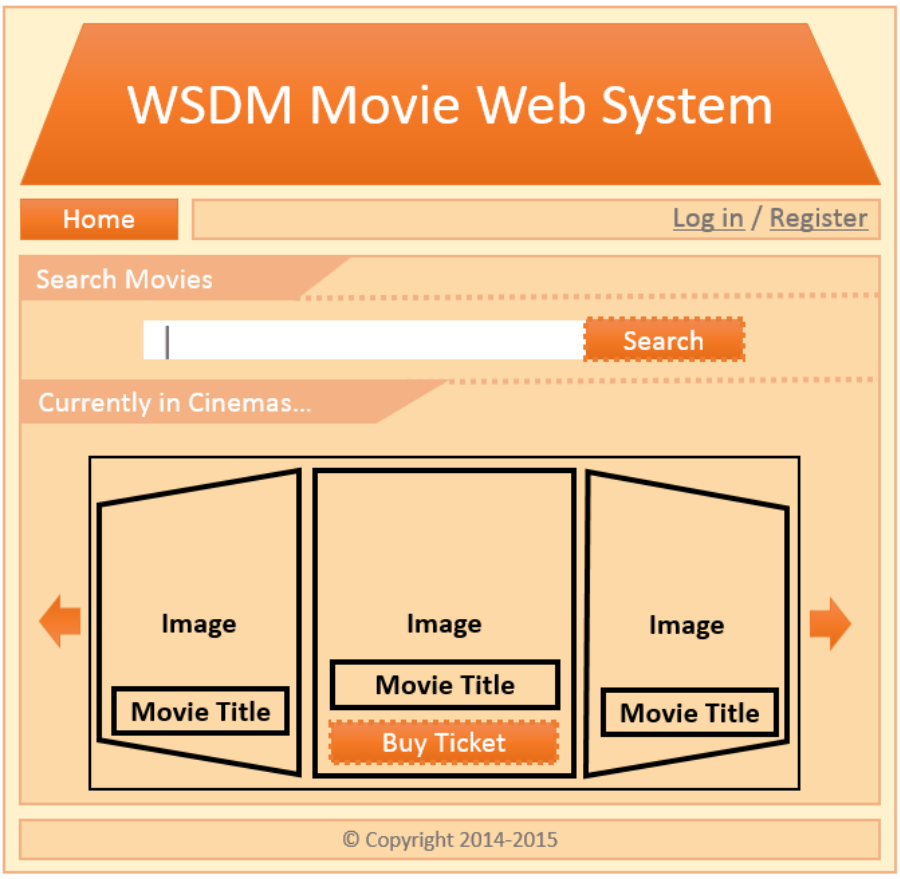


Figure 5.20: Page model for the homepage

6 Conclusion

The objective of this research was to investigate how the Interaction Flow Modelling Language could be fit into the Web Semantics Design Method. This is relevant because the use of IFML would be beneficial to WSDM, as it is a standardised language that covers a range of objectives which to some extent overlap with those of WSDM. Using a standardised language would increase the acceptance and accessibility of WSDM as a web design method. A second objective of the investigation was to assess whether IFML would be able to replace more than one of WSDM's modelling languages in order to reduce the effort to learn and understand WSDM.

6.1 Contributions

Each phase and sub-phase of WSDM was analysed to assess whether IFML could be used for it. This led to our first contribution: the definition of a so-called conceptual IFML. This was needed because standard IFML imposes certain constraints on the user interface which do not belong to a purely conceptual design. Conceptual IFML was used to replace the navigational modelling language in the Navigational Design sub-phase.

We also provided an algorithm to transform the CTT task trees modelled in the Task Modelling sub-phase directly into conceptual IFML. This is our second contribution. We argue that this transformation can be done fully

automatically.

In order to perform the transformation, we have adjusted the Task Modelling phase to include extra CTT trees, named "audience class trees", which capture each of the requirement task trees in a single tree per audience class. This adjustment contributes to the Task Modelling sub-phase in the sense that it now models the task structure for the entire system. This is our third contribution.

Next, we used IFML also to replace the Site Structure Design sub-phase by an Interaction Modelling sub-phase. This is not an extra step, on the contrary, it is a reorganisation step in which the developer transforms the conceptual IFML diagram into a standard IFML diagram. This inherently leads to a site structure design, as a standard IFML diagram defines the pages of the web system. Moreover, it simplifies the Presentation Design sub-phase, as the IFML model already specifies which user interface components need to be available to the user and what events can occur to them. This is the fourth contribution.

Hence, the overall contribution of this thesis is an adapted version of WSDM in which different WSDM-specific modelling languages have been replaced by IFML. The navigational modelling language and its site structure design extension were replaced by standard IFML and the very closely-related conceptual IFML. This not only reduces the number of different modelling languages used in WSDM, but also opens the possibility to use tools that are or will be developed for IFML.

6.2 Discussion

Due to the diverse goals of the different modelling languages in WSDM, the integration of IFML did not reduce the number of different languages in WSDM significantly: two languages were replaced by two variants of IFML, conceptual IFML and standard IFML.

Another criticism could be the abundance of activation expressions in a conceptual IFML model. These might blur the structure of the model and look complex. However, only for certain types of transformation patterns these activation expressions grow large and a single IFML construct cannot have more than one. The expressions are an essential part of a conceptual IFML model and indicate the semantics of the model accurately. Moreover, they are valid standard IFML constructs, and therefore they should be easy to understand by a developer.

6.3 Future Work

Future work could be dedicated to the implementation of a tool that automates the translation from CTT tasks to a conceptual IFML model. This tool could present the resulting model to the developer and provide him with a workbench to transform it to standard IFML.

This standard IFML can be input for another functionality of the tool that could transform it into the basic structures for the page models which need to be defined in the Presentation Design sub-phase.

Ideally, the tool could export the final IFML model and the partial page models in a format understandable by the WebRatio tool. Once imported, the IFML model could be enhanced with WebRatio-specific constructs and the partial page models could be elaborated through a WebRatio *style* project. At last, the entire web system could be generated by WebRatio.

Bibliography

Acerbis, R., Bongio, A., Brambilla, M., & Butti, S. (2007). WebRatio 5: An Eclipse-based CASE tool for engineering web applications. In L. Baresi, P. Fraternali, & G. Houben (Eds.), *Proceedings of Web Engineering: International Conference (ICWE2007)*, July 16-20, 2007, Como, Italy (pp. 501–505).

Acerbis, R., Bongio, A., Butti, S., Ceri, S., Ciapessoni, F., Conserva, C., ... Toffetti, G. (2004). WebRatio, an innovative technology for web application development. In N. Koch, P. Fraternali, & M. Wirsing (Eds.), *Proceedings of Web Engineering: International Conference (ICWE2004)*, July 26-30, 2004, Munich, Germany (pp. 613–614).

Beck, K. (1999). Embracing change with extreme programming. *IEEE Computer*, *32*(10), 70-77.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American*, *284*, 34–43.

Brambilla, M., & Fraternali, P. (2014). *Interaction Flow Modeling Language: Model-driven UI engineering of Web and mobile apps with IFML*. Waltham, MA: Morgan Kaufmann.

Brambilla, M., Mauri, A., & Umuhoza, E. (2014). Extending the Interaction Flow Modeling Language (IFML) for model driven development of mobile applications front end. In I. Awan, M. Younas, X. Franch, & C. Quer (Eds.), *Proceedings of the Mobile Web Information Systems Conference (MobiWIS2014)*, August 27-29, 2014, Barcelona, Spain (pp. 176–191).

Casteleyn, S., & De Troyer, O. (2002). Structuring web sites using audience class hierarchies. In H. Arisawa, Y. Kambayashi, V. Kumar, H. C. Mayr, & I. Hunt (Eds.), *Conceptual Modeling for New Information Systems Technologies, ER 2001 Workshops, HUMACS, DASWIS, ECOMO, and DAMA*, November 27–30, 2001, Yokohama, Japan (pp. 198–211).

Ceri, S., Fraternali, P., & Bongio, A. (2000). Web Modeling Language (WebML): A modeling language for designing web sites. *Computer Networks*, *33*(1-6), 137–157.

De Troyer, O. (2001). Audience-driven web design. In M. Rossi & K. Siau (Eds.), *Information modeling in the new millennium* (pp. 442–462). London: Idea Group Publishing.

De Troyer, O., Casteleyn, S., & Plessers, P. (2005). Using ORM to model web systems. In R. Meersman et al. (Eds.), *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops, International Workshop on Object-Role Modeling (ORM'05)*, October 31 - November 4, 2005, Agia Napa, Cyprus (pp. 700–709).

De Troyer, O., Casteleyn, S., & Plessers, P. (2008). WSDM: Web Semantics Design Method. In G. Rossi, O. Pastor, D. Schwabe, & L. Olsina (Eds.), *Web engineering: Modelling and implementing web applications* (pp. 303–351). London: Springer.

De Troyer, O., & Leune, C. J. (1998). WSDM: A user centered design method for web sites. In *Proceedings of the Seventh International World Wide Web Conference (WWW7)*, April 14-18, 1998, Brisbane, Australia (pp. 85–94).

Halpin, T., & Morgan, T. (2008). *Information modeling and relational databases*. Burlington, MA: Morgan Kaufmann.

Kleppe, A. G., Warmer, J., & Bast, W. (2003). *MDA explained: The model driven architecture: Practice and promise*. Boston, MA: Addison-Wesley.

Koch, N., & Hennicker, R. (2000). A UML-based methodology for hypermedia design. In A. Evans, S. Kent, & B. Selic (Eds.), *Proceedings of UML 2000: The Unified Modeling Language. Advancing the Standard: Third International Conference*, October 2-6, 2000, York, UK (pp. 410–424).

Lima, F., & Schwabe, D. (2003). Modeling applications for the semantic web. In J. M. C. Lovelle, B. M. G. Rodríguez, J. E. L. Gayo, M. del Puerto Paule Ruiz, & L. J. Aguilar (Eds.), *Proceedings of Web Engineering: International Conference (ICWE2003)*, July 14-18, 2003, Oviedo, Spain (pp. 417–426).

Object Management Group. (2015). *IFML 1.0 specification*. Retrieved on June 2, 2015, from `http://www.omg.org/spec/IFML/1.0/PDF/`

Pastor, O., Fons, J., Pelechano, V., & Abrahao, S. (2006). Conceptual modelling of web applications: the OOWS approach. In E. Mendes & N. Mosley (Eds.), *Web engineering* (pp. 277–302). Berlin: Springer.

Paterno, F. (2000). *Model-based design and evaluation of interactive applications* (R. J. Paul, P. J. Thomas, & J. Kuljis, Eds.). London: Springer-Verlag.

Paterno, F. (2003). ConcurTaskTrees: An engineered notation for task models. In D. Diaper & N. A. Stanton (Eds.), *The handbook of task*

analysis for human-computer interaction (pp. 483–503). Mahwah, NJ: Lawrence Erlbaum Associates.

Paterno, F., Mancini, C., & Meniconi, S. (1997). ConcurTaskTrees: a diagrammatic notation for specifying task models. In S. Howard, J. Hammond, & G. Lindgaard (Eds.), *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction (INTERACT97)*, July 14-18, 1997, Sydney, Australia (pp. 362–369).

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77.

Plessers, P., Casteleyn, S., & De Troyer, O. (2005). Semantic web development with WSDM. In S. Handschuh, T. Declerck, & M.-R. Koivunen (Eds.), *Proceedings of the 5th International Workshop on Knowledge Markup and Semantic Annotation (SemAnnot2005)*, November 7, 2005, Galway, Ireland (pp. 1–12).

Rodriguez-Echeverria, R., Pavón, V. M., Macías, F., Conejero, J. M., Clemente, P. J., & Sánchez-Figueroa, F. (2014). IFML-based model-driven front-end modernization. In V. Strahonja et al. (Eds.), *Proceedings of the Information Systems Development Conference (ISD2014)*, September 2-4, 2014, Varaždin, Croatia (pp. 226–233).

Schwabe, D., & Rossi, G. (1995). The object-oriented hypermedia design model. *Communications of the ACM*, *38*(8), 45–46.

Schwabe, D., & Rossi, G. (1998). An object oriented approach to web-based applications design. *Theory and Practice of Object Systems*, *4*(4), 207–225.