

A Requirements Engineering Approach for the Development of Web Applications

Pedro J. Valderas Aranda

Department of Information Systems and Computation
Technical University of Valencia



UNIVERSIDAD
POLITECNICA
DE VALENCIA

A thesis submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy in Computer
Science

Supervisor: Dr. Vicente Pelechano Ferragud

November 2007

Table of Contents

1	Introduction.....	1
1.1	Motivation.....	2
1.2	The Problem Statement.....	4
1.3	Main Contributions	5
1.4	Scope.....	7
1.5	Research Methodology	7
1.6	The Context of the Thesis	9
1.7	Thesis Structure.....	9
2	State of the Art	13
2.1	Requirements Specification in the context of Web engineering	13
2.2	Study of Web Engineering Methods.....	15
2.2.1	OOHDM: Object Oriented Hypermedia Design Model	15
2.2.2	WSDM: Web Site Design Method.....	18
2.2.3	SOHDM: Scenario-based Object-Oriented Hypermedia Design Methodology	21
2.2.4	UWE: UML-based Web Engineering	23
2.2.5	Building Web Applications with UML.....	26
2.2.6	WebML: Web Modelling Language	29
2.2.7	OO-H: Object-Oriented Hypermedia Method.....	31
2.2.8	W2000.....	33
2.2.9	NDT: Navigational Development Techniques.....	35
2.2.10	Proposals of new Development Processes for Web Applications.....	37
2.2.11	Other Studied Approaches	40
2.3	Conclusions.....	42
3	A Web Application Development Process.....	45
3.1	Describing Software Processes. The SPEM Notation.....	46
3.2	The OOWS development Process with RE support.....	48
3.3	Improving the Requirements Analysis Activity throughout Prototyping ...	51
3.3.1	An Iterative and Prototyping RE Process for Web applications	53
3.4	Conclusions.....	58

4. A Task-Based Requirements Model for Specifying Web Applications.....	59
4.1 Tasks. Background and Motivation	60
4.1.1 The Concept of Task	60
4.1.2 Task Analysis in Software Engineering	62
4.2 Requirements Model Architecture.....	63
4.2.1 Statement of Purpose	65
4.2.2 User Task Description.....	66
4.2.3 System Data Description.....	92
4.3 A RE tool for Creating Task-based Requirements Models.....	99
4.3.1 The Plug-ins used to develop the RE tool	100
4.3.2 The RE tool user interface.....	103
4.4 Conclusions.....	108
5 Requirements Traceability.....	111
5.1 Traceability in Requirements Engineering.....	112
5.1.1 Other Traceability Classifications.....	114
5.2 Defining Traceability Rules: A General View	115
5.3 Traceability Rules Catalogue	116
5.3.1 Structural Model Traceability Rules	117
5.3.2 Navigational Model.....	134
5.4 Conclusions.....	177
6 Applying Traceability Rules through Model-to-Model Transformations ..	179
6.1 Model-to-Model Transformations.....	180
6.1.1 Graph Transformations as Model-to-Model Transformation Technique	183
6.2 From Requirements Models to Conceptual Models throughout Graph Transformations.....	184
6.2.1 Graph Transformations in a Nutshell.....	184
6.2.2 Defining Graph Transformations	185
6.2.2 A Tool for Defining Graph Grammars.....	198
6.3 Automatic Application of Graph-Grammar-based Model-to-Model Transformations.....	202
6.4 Conclusions.....	208
7 Prototyping Requirements	211
7.1 Tool Support: A “Big Picture”	212
7.2 From Requirements to Prototypes. Step by Step.	214
7.2.1 Step 1: A task based requirements model	214

7.2.2 Step 2: A graph-based representation of the task-based requirements model	215
7.2.3 Step 3: Transforming the Graph	216
7.2.4 Step 4: OOWS models in the proper XML specification	218
7.2.5 Step 5: Generation of Web Application prototypes	219
7.3 Supporting Requirements with the Generated Web Application Prototypes	221
7.3.1 Supporting the Task Taxonomy	221
7.3.2 Adding CDs: Lists of Music Categories and CDs	222
7.3.3 Adding CDs: Descriptions of CDs and Artists	224
7.4 Conclusions	226
8 Conclusions	227
8.1 Main Contributions	227
8.2 Current and Further Work	229
8.3 Publications	231
8.4 A Final Reflection	234
<i>Appendix A: Meta-Model of the Task-based Requirements Model</i>	<i>255</i>
<i>Appendix B: The OOWS Approach in a Nutshell</i>	<i>267</i>
<i>Appendix C: Some Essentials of Graph Transformations</i>	<i>287</i>
<i>Appendix D: A CASE Study: An Amazon-like E-Commerce Application</i>	<i>303</i>

List of Figures

Figure 1.1 Research methodology followed in this thesis	8
Figure 2.1 Example of requirements specification in OOHDM	17
Figure 2.2 Example of requirements specification in WSDM	20
Figure 2.3 Example of requirements specification in SOHDM	23
Figure 2.4 Example of requirements specification in UWE	26
Figure 2.5 Example of requirements specification in the Conallens' approach.....	28
Figure 2.6 Example of requirements specification in WebML.....	31
Figure 2.7 Example of requirements specification in OOH.....	33
Figure 2.8 Example of requirements specification in W2000.....	34
Figure 2.9 Example of requirements specification in NDT	36
Figure 3.1 SPEM Notation.....	47
Figure 3.2 The Extended OOWS development process.....	50
Figure 3.3 The prototype construction process	53
Figure 3.4 A RE process for Web applications.....	56
Figure 4.1 Requirements Model Architecture.....	64
Figure 4.2 Graphical notations for tasks	68
Figure 4.3 Example of structural refinement	69
Figure 4.4 Example of temporal refinement	72
Figure 4.5 The task taxonomy of the Amazon Example.....	74
Figure 4.6 Information related to the performance of <i>Add CD</i>	76
Figure 4.7 A first example of a task performance description	78
Figure 4.8 System Action representations	79
Figure 4.9 IP representations	81
Figure 4.10 Example of user action: selection of information.....	83
Figure 4.11 Example of double arrowed arc.....	83
Figure 4.12 Example of operation activation	84
Figure 4.13 Example of user action: introduction of information to perform a system action.....	85
Figure 4.14 Example of user action: activation of operation + introduction of information.....	85
Figure 4.15 Example of sequence of system actions	86
Figure 4.16 Example of transfer of information	87
Figure 4.17 Example of conditioned flow.....	88

Figure 4.18 Example of initial step	89
Figure 4.19 Example of final step	89
Figure 4.20 <i>Add CD</i> Elementary Task.	90
Figure 4.21 <i>Inspect Shopping Cart</i> Elementary Task.	91
Figure 4.22 <i>Checkout</i> Elementary Task.	92
Figure 4.23 Information template associated to the entity CD	94
Figure 4.24 Information template associated to the entity Artist	95
Figure 4.25 Information template associated to the entity Product	95
Figure 4.26 Exchanged Data Template associated to the IP Output(CD,*)	96
Figure 4.27 Exchanged Data Template associated to the IP Output(Product,1)	98
Figure 4.28 Exchanged Data Template associated to the IP Output(Item,*)	99
Figure 4.29 The MVC pattern with GEF	101
Figure 4.30 Steps for developing the graphical editor	102
Figure 4.31 Partial view of the task taxonomy XMI serialization	103
Figure 4.32 Eclipse-based user interface of the graphical editor	104
Figure 4.33 Modelling zone, tool bar and property editor for creating task taxonomies	105
Figure 4.34 Modelling zone, tool bar and property editor for creating task performance descriptions	106
Figure 4.35 Modelling zone and tool bar and for creating information templates....	107
Figure 4.36 Modelling zone and property editor for creating exchanged data templates	107
Figure 5.1 Bidirectional Traceability	113
Figure 5.2 Example of application of Rule S1	119
Figure 5.3 Another example of application of Rule S1	119
Figure 5.4 Example of application of Rule S2	120
Figure 5.5 Another example of application of Rule S2	121
Figure 5.6 Example of application of Rule S3a	122
Figure 5.7 Example of application of Rule S3a and Rule S3b	123
Figure 5.8 Example of application of Rule S4	125
Figure 5.9 Example of application of Rule S5	126
Figure 5.10 Examples of application of Rule S6	127
Figure 5.11 Example of application of Rule S7	128
Figure 5.12 Example of application of Rule S8a and S8b	130
Figure 5.13 Partial version of the structural model of the Amazon example	132
Figure 5.14 Derivation of the OOWS Navigational Model	134
Figure 5.15 Example of application of Rule N1a	136
Figure 5.16 Example of application of Rule N1b	137
Figure 5.17 Example of application of Rule N2	138
Figure 5.18 Another example of application of Rule N2	139
Figure 5.19 Example of application of Rules N3a and N3b	141
Figure 5.20 Example of application of Rules N3c	143
Figure 5.21 Example of application of Rules N3a and N3d	145
Figure 5.22 Example of application of Rule N3e	146

Figure 5.23 Example of application of Rule N3f.....	147
Figure 5.24 Example of application of Rules N4a and N4b	149
Figure 5.25 Example of application of Rules N4d and N4g	152
Figure 5.26 Example of application of Rules N5a and N5b	154
Figure 5.27 Implicitly defined navigational relationships	155
Figure 5.28 Example of application of Rule N7c	164
Figure 5.29 Example of application of Rule N8	166
Figure 5.30 Example of application of Rule N11	170
Figure 5.31 Partial version of the navigational map of the Amazon example	173
Figure 5.32 Partial version of the navigational contexts of the Amazon example....	174
Figure 6.1 A transformation system.....	185
Figure 6.2 Example of a type graph.....	187
Figure 6.3 Relation between type graphs and graphs	188
Figure 6.4 The type graph used in this thesis.....	190
Figure 6.5 Partial view of a task based requirements model represented as a graph	192
Figure 6.6 Graph transformation rule for traceability rule N1a	193
Figure 6.7 Graph transformation rule for traceability rule N2.....	194
Figure 6.8. Graph transformation rule for traceability rule N4a.....	195
Figure 6.9 Graph transformation rule for traceability rule N11	196
Figure 6.10 Graph transformation rule for traceability rule N4b.....	197
Figure 6.11 Graph transformation rule for traceability rule N4c	197
Figure 6.12 Graph transformation rule for traceability rule N5a	198
Figure 6.13 AGG user interface.....	200
Figure 6.14 AGG user interface for editing attributes	200
Figure 6.15 Partial view of an XML AGG graph.	201
Figure 6.16 Task2Graph user interface.....	203
Figure 6.17 Graph representation of a task-based requirements model.....	204
Figure 6.18 Graph that represents an OOWS Structural Model	205
Figure 6.19 Graph that represents an OOWS Navigational Model.....	206
Figure 6.20 Graph2OOWS user interface.....	206
Figure 6.21 Navigational model derived from a graph-based description	207
Figure 6.22 Tool-supported strategy for automating model-to-model transformations	208
Figure 7.1 Tool Support in RE activities	213
Figure 7.2 Step 1: Creation of the task-based requirements model	215
Figure 7.3 Step 2: Task-based requirements models as graphs.....	216
Figure 7.4 Step 3: OOWS models as graphs.....	217
Figure 7.5 Step 4: OOWS models in the proper format.....	218
Figure 7.6 Step 5: Web application prototype generation.....	219
Figure 7.7 Generated files for the Amazon example	220
Figure 7.8 PHP files interconnected according to the navigational model	221
Figure 7.9 Home page for the Amazon Example.....	222
Figure 7.10 Web pages that provide lists of Music Categories and CDs.....	223

Figure 7.11 Web pages that provide descriptions about a CD and an artist225

Abstract

One of the most important problems that were presented to be solved when Web Engineering emerged was the insufficient requirements specification techniques that exist for Web applications.

Although a significant number of proposals provide methodological solutions for developing Web applications, they mainly focus on defining Web applications from conceptual models, and do not pay much attention to the specification of requirements. Furthermore, traditional techniques for specifying requirements are not appropriate to support distinctive characteristics of Web applications such as *Navigation*

In this thesis, we present a Requirements Engineering approach for specifying Web applications requirements. This approach includes mechanisms based on the task metaphor for specifying not only requirements related to the structural and behavior aspect of Web applications but also those related to the navigational aspects.

However, requirements specifications alone are of little use if we do not translate them into the proper software artefacts. This is a classical problem that the Software Engineering community has been trying to solve from its beginning: how to go from the *problem space* (user requirements) to the *solution space* (design and implementation) with a sound methodological guidance.

In this thesis, we present a tool-supported strategy based on graph transformations that allows us to automatically perform model-to-model transformations between task-based requirements specifications and Web conceptual schemas. Furthermore, this strategy has been integrated with a Web engineering method that provides us with code generation capabilities. This integration allows us to provide a mechanism to automatically generate Web application prototypes from requirements specification.

Resumen

Uno de los problemas más importantes que se propuso solucionar cuando apareció la Ingeniería Web fue la carencia de técnicas para la especificación de requisitos de aplicaciones Web.

Aunque se han presentado diversas propuestas que proporcionan soporte metodológico al desarrollo de aplicaciones Web, la mayoría de ellas se centran básicamente en definir modelos conceptuales que permiten representar de forma abstracta una aplicación Web; las actividades relacionadas con la especificación de requisitos son vagamente tratadas por estas propuestas. Además, las técnicas tradicionales para la especificación de requisitos no proporcionan un soporte adecuado para considerar características propias de las aplicaciones Web como la Navegación.

En esta tesis, se presenta una aproximación de Ingeniería de Requisitos para especificar los requisitos de las aplicaciones Web. Esta aproximación incluye mecanismos basados en la metáfora de tarea para especificar no sólo los requisitos relacionados con aspectos estructurales y de comportamiento de una aplicación Web sino también los requisitos relacionados con aspectos navegacionales.

Sin embargo, una especificación de requisitos es poco útil si no somos capaces de transformarla en los artefactos software adecuados. Este es un problema clásico que la comunidad de Ingeniería del Software ha tratado de resolver desde sus inicios: cómo pasar del *espacio del problema* (requisitos de usuario) al espacio de la solución (diseño e implementación) siguiendo una guía metodológica clara y precisa.

En esta tesis, se presenta una estrategia que, basándose en transformaciones de grafos, y estando soportada por un conjunto de herramientas, nos permite realizar de forma automática transformaciones entre especificaciones de requisitos basadas en tareas y esquemas conceptuales Web. Además, esta estrategia se ha integrado con un método de Ingeniería Web con capacidades de generación automática de código. Esta integración nos permite proporcionar un mecanismo para generar de forma automática prototipos de aplicaciones Web a partir de especificaciones de requisitos.

Resum

Un dels problemes més importants que es va proposar solucionar quan va aparèixer l'Enginyeria Web va ser la manca de tècniques per a l'especificació de requisits d'aplicacions Web.

Encara que s'han presentat diverses propostes que proporcionen suport metodològic al desenvolupament d'aplicacions Web, la majoria d'elles es centren bàsicament en definir models conceptuals que permeten representar de forma abstracta una aplicació Web; les activitats relacionades amb l'especificació de requisits són vagament tractades per aquestes propostes. A més, les tècniques tradicionals per a l'especificació de requisits no proporcionen un suport adequat per a considerar característiques pròpies de les aplicacions Web com pot ser la Navegació.

En aquesta tesi, es presenta una aproximació d'Enginyeria de Requisits per a especificar els requisits de les aplicacions Web. Aquesta aproximació inclou mecanismes basats en la metàfora de tasca per a especificar no sols els requisits relacionats amb aspectes estructurals i de comportament d'una aplicació Web, sinó també els requisits relacionats amb aspectes navegacionals.

No obstant això, una especificació de requisits és poc útil si no som capaços de transformar-la en els artefactes de programari corresponents. Aquest és un problema clàssic que la comunitat d'Enginyeria del Programari ha tractat de resoldre des dels seus inicis: com passar de l'espai del problema (requisits d'usuari) a l'espai de la solució (disseny i implementació) seguint una guia metodològica clara i precisa.

En aquesta tesi, es presenta una estratègia que, basant-se en transformacions de grafs, i estant suportada per un conjunt d'eines, ens permet realitzar de forma automàtica transformacions entre especificacions de requisits basades en tasques i esquemes conceptuals Web. A més, aquesta estratègia s'ha integrat amb un mètode d'enginyeria Web amb capacitats de generació automàtica de codi. Aquesta integració ens permet proporcionar un mecanisme per a generar de forma automàtica prototips d'aplicacions Web a partir d'especificacions de requisits.

Chapter 1

*“Man is the only animal that
can be skinned more than once.”*

Jimmy Durante.
(American Comedian, Pianist and Singer, 1893-1980)

1 Introduction

The work presented in this thesis is related to two engineering paradigms: *Requirements Engineering* (RE) and *Web Engineering* (WE). RE can be defined as “the area of software engineering that focuses on the RE process which involves understanding customer needs and expectations (requirements elicitation), requirements analysis and specification, requirements prioritization, requirements derivation, partitioning and allocation, requirements tracing, requirements management, requirements verification, and requirements validation” [Young 2003]. WE can be defined as “the establishment and use of sound scientific, engineering and management principles and disciplined and systematic approaches to the successful development, deployment and maintenance of high quality Web-based applications” [Murugesan et al. 1999].

Web-based applications are software systems delivered to users through an Internet technology. Other terms have been used in the literature to refer to systems of this kind. Web sites, Web-based systems, or Web applications are just some examples. This thesis uses the term *Web application* to represent all the variations. In order to refer to non-Web-based applications, the term *traditional software* is used.

In this thesis, we present a RE approach for Web applications. This approach is introduced to support the specification of Web application requirements as well as to facilitate the creation of Web application conceptual schemas from requirements

1. Introduction

specifications. To do this, we present a novel technique based on the concept of task for specifying Web applications requirements. Furthermore, we present a methodological guideline to transform task-based requirements specifications into Web application conceptual schemas. This guideline is complemented with a tool-supported model-to-model transformation strategy that allows us to automate its application by using graph transformations. Additionally, we integrate this model-to-model transformation strategy with a Web engineering method with code generation capabilities. This aspect allows us to define mechanisms for automatically obtaining Web application prototypes from task-based requirements specifications.

The rest of this chapter is organized as follows: Section 1.1 explains the purpose of this thesis. In section 1.2, the problem that this thesis resolves is stated in detail. Next, the main contributions of this thesis are summarized in Section 1.3. The scope of the thesis is introduced in Section 1.4. The research methodology that we have followed is presented in Section 1.5. Section 1.6 explains the context in which the work of this thesis has been performed. Finally, Section 1.4 describes the structure of the thesis.

1.1 Motivation

It is well known that effective and efficient requirements engineering activities are absolutely essential if software systems are to meet the expectations of their customers and users are to be delivered on time and within budget [Al-Rawas et al. 1996]. However, one of the most important problems that was presented to be solved when Web Engineering was introduced in the late nineties [Murugesan et al. 1999] [Deshpande & Hansen 2001] was that existing requirements specification techniques were insufficient for Web applications.

In the context of the Requirements Engineering community, several approaches for the specification of requirements have been presented during the last three decades. Some examples are: Constantine and Lockwood (1999), Jaaksi (1998), Leite *et al.* (1997), Durán *et al.* (1999) or Rosenberg and Scott (1999). These approaches have done excellent work in specifying requirements related to the structure and the behaviour of a software system. Other approaches are Chung *et al.* (1999), Cysneiros *et al.* (1992), or Botella *et al.* (2001). These approaches have successfully focused their efforts on defining mechanisms for the description of non-functional requirements related to aspects such as the performance, the reusability or the reliability of a software system. However, all these approaches are not enough for the specification of Web application requirements. Web applications are developed over the World Wide Web (WWW) paradigm [Berners-Lee *et al.* 1994] in which an aspect that has been poorly considered in traditional software development has become critical for Web applications: Navigation [Cachero & Koch et al. 2002a] [Schwabe *et al.* 1996] [Ceri et al. 2000].

1. Introduction

Navigation is encouraged in Web applications because the focus of Web applications is extensively based on communication [Greenspun 1999]. Since the initial goal of the WWW is based on its role as an information medium¹, many Web applications are fully or partially developed as magazines or brochures, and their development mainly involves capturing and organizing a complex information domain and making that domain accessible to users. The information is organized in a structure made up of nodes, links, and anchors (according to the hypertext paradigm [Conklin 1987]), which allows users to *navigate* throughout the information in a non-linear way.

The RE activities for these publishing-oriented Web applications require taking decisions that typically rely on an ‘editor’ rather than on an ‘engineer’. Determining the organization for this information (i.e. the *navigational structure*) is similar to deciding the structure of a magazine or a brochure. Consider for instance large Web applications such as Amazon², Yahoo³ or Ebay⁴ where a major part of these applications is oriented specifically to provide users with product information. In the words of Philip Greenspun [Greenspun 1999]: “When you put a magazine-like site, you are publishing. Virtually all of the important decisions that you must make are publishing decisions. Eventually you will have to select technology to support those decisions, but that is a detail”.

In the context of Web engineering, Navigation is considered to be a first-order citizen. Several approaches such as OOHDM [Schwabe *et al.* 1996], WebML [Ceri *et al.* 2000], UWE [Koch 2000], WSDM [De Troyer & Leune 1998], W2000 [Baresi *et al.* 2001], SOHDM [Lee *et al.* 1998], RNA [Yoo & Bieber 1998] or OOH [Cachero 2003] have been presented in order to support the development of Web applications. They provide different development processes in which a navigational model plays a main role. This model has been introduced to properly handle Navigation. However, this model is focused on describing Navigation at the conceptual level. Little support is provided for describing Navigation at the requirements level [Escalona & Koch 2004].

Most Web engineering methods prescribe the use of traditional requirements techniques for specifying Web application requirements. However, several works such as England & Finney (1999), Burdman (1999), Overmyer (2000) and Lowe (2003) have stated that new RE techniques are required for specifying the requirements of Web applications since RE techniques for traditional software are not appropriate for supporting distinctive characteristics of Web applications such as Navigation. In relation with this, we can see how Web Engineering researchers’ initiatives such as

¹ The WWW was originally created for the purpose of sharing scientific information among a few scientists

² [Http://www.amazon.com](http://www.amazon.com)

³ [Http://www.yahoo.com](http://www.yahoo.com)

⁴ [Http:// www.ebay.com](http://www.ebay.com)

1. Introduction

MDWnet [Vallecillo *et al.* 2007] are starting to consider the field of RE for Web application as a new concern to be addressed.

Keeping this lack of RE techniques for specifying Web application requirements in mind, it is not surprising that a survey done by the Cutter Consortium [Cutter 2000] discovered that the top problem areas of large-scale Web applications projects were:

- Project schedule delays (79%)
- Budget overruns (63%)
- Lack of required functionality (53%)
- Poor quality of deliverables (52%)

These problems are very similar to those that the RE community has been trying to solve in the last three decades for the development of traditional software [GAO 1979] [TSG 1995]. These problems were mainly produced by a lack of user information and an incomplete and variable requirements specification.

In a survey done by Macdonald & Welland (2001), Web application developers claim that most of their Web development projects are running over budget and overtime because of basically two reasons: problems in capturing requirements and poor communication between developers and their clients.

It is clear from the above discussion that RE techniques for specifying Web applications requirements are needed. However, requirements specifications alone are of little use if we do not translate them into the proper software artefacts. This is a classical problem that the software engineering community has been trying to solve from its beginning: how to go from the *problem space* (user requirements) to the *solution space* (design and implementation) with a sound methodological guidance.

From the perspective of a relevant software development approach such as Model-Driven Development (MDD) [Mellor *et al.* 2003], this problem is re-formulated as follows: how to go from a requirements model to a conceptual model that satisfies every requirement and allows us to derive the proper code later. In [Insfrán 2003], this problem is handled by introducing traceability rules which help analysts to create conceptual models from requirements models. However, this work is only focused on the structural and behavioural aspects of software systems; Navigation is not properly considered.

1.2 The Problem Statement

The development of Web applications is not a closed research topic. The above discussion indicates that some problems still need to be considered. The work that has

1. Introduction

been done in this thesis is an attempt to improve the development of Web applications by considering these problems, which can be stated by the following three research questions:

Research Question 1. How should Web applications requirements be specified in order to properly consider not only the structural and behavioural aspects of Web applications but also the navigational aspect?

Research Question 2. How should a methodological guidance be defined in order to facilitate the construction of Web application conceptual models from requirements specifications?

Research Question 3. How should RE activities be performed within the development process of Web applications?

1.3 Main Contributions

The main contributions of this thesis have been developed to answer the three research questions presented above:

- 1 We present a **RE approach for specifying Web applications requirements**. This approach allows us to properly consider distinctive characteristics of Web applications such as Navigation at the requirements level. This approach is based on the task metaphor, which is widely accepted for the capture of traditional requirements [Lauesen 2003]. However, it has been extended to properly capture the navigational aspect of Web applications. We extend traditional task descriptions by introducing information about the interaction between the user and the system.

This contribution is obtained by achieving the following goals:

- 1.1 Studying the different approaches that are proposed to capture Web application requirements, by paying special attention to the way in which Navigation is captured and by analysing their main limitations.
- 1.2 Identifying the main abstractions that characterize Web applications at the requirements level and proposing a Requirements Model to represent these abstractions. As introduced above, this Requirements Model is based on the task metaphor, which is extended with aspects related to the interaction between the user and the system.

1. Introduction

- 1.3 Developing a RE tool for supporting the visual creation and management of task-based requirements models. This tool is developed by using the facilities provided by the Eclipse development platform [Eclipse].
- 2 We introduce a **tool-supported strategy in order to automatically derive Web application conceptual models from requirements specifications**. In this thesis, we use the conceptual model provided by the Web Engineering method OOWS [Fons *et al.* 2003] in order to represent Web applications at the conceptual level. A technique based on graph transformation is used to automate the derivation of OOWS conceptual models from Web application requirements specifications.

This contribution is obtained by achieving the following goals:

- 2.1 Identifying the correspondences between the abstractions of the task-based requirements model and the primitives of the OOWS conceptual model.
- 2.2 Defining these correspondences by means of graph transformation rules.
- 2.3 Providing tools to automate the application of graph transformation rules. In order to apply graph transformation rules we propose the use of the AGG tool [AGG]. Since this tool works only with graphs (no models), two more tools have been developed to transform (1) task-based requirements models into graphs and (2) graphs into OOWS conceptual models.
- 3 We define **the OOWS development process by using a process description technique such as SPEM** [SPEM 2002] and extend it to support RE activities for Web applications. To do this, an **iterative and prototyping RE process for Web applications** is proposed. This RE process proposes the iterative performance of different activities in order to handle Web application requirements. Within these activities, Web application prototypes are proposed to be automatically generated from requirements models. These prototypes constitute a valuable tool for helping customers to validate requirements.

A key factor in the generation of prototypes from requirements models is the OOWS code generation strategy. It allows us to automatically generate Web application prototypes from its conceptual model. If we consider that one of the contributions of this thesis is a tool-supported strategy to automatically obtain OOWS conceptual models from task-based requirements models, we are implicitly obtaining Web application prototypes from requirements models automatically.

Finally, it is worth noting that the OOWS code generation strategy (and the possibility of generating prototypes during RE activities with which this strategy provides us) is one of the main reasons for using the OOWS method in

1. Introduction

this thesis. Another reason for this choice is the extensive knowledge that the research group in which this thesis has been developed has about this method. This aspect has facilitated an exhaustive analysis of the best way to support requirements model abstractions with the OOWS conceptual primitives.

1.4 Scope

It is clear that there are several families of Web applications, which may be classified according to different criteria such as their domain (E-commerce, healthcare, educational, corporate, etc.) or their goals (trading goods, community building, informing, entertaining, etc.). The scope of this thesis is delineated by means of the Web application classification proposed in [Ginige 2001].

This thesis provides support to those Web application that are classified into the following categories:

- **Informational:** For instance, online newspapers, product catalogues, newsletters, service manuals, online classifieds, or online electronic books.
- **Transactional:** For instance, electronic shopping, ordering goods and services, online banking.
- **Web Portals:** For instance, electronic shopping malls or online intermediaries.

1.5 Research Methodology

In order to perform the work of this thesis, we have carried out a research project following the design methodology for performing research in information systems as described by [March & Smith 1995] and [Vaishnavi & Kuechler 2004]. Design research involves the analysis of the use and performance of designed artefacts to understand, explain and, very frequently, to improve on the behaviour of aspects of Information Systems [Vaishnavi & Kuechler 2004].

The design cycle consists of 5 process steps: (1) awareness of the problem, (2) suggestion, (3) development, (4) evaluation, and (5) conclusion. The design cycle is an iterative process; knowledge produced in the process by constructing and evaluating new artefacts is used as input for a better awareness of the problem.

1. Introduction

Following the cycle defined in the design research methodology, we started with the awareness of the problem (see Figure 1.1): We identified the problem to be resolved and we stated it clearly.

Next, we performed the second step which is comprised of the suggestion of a solution to the problem, and comparing the improvements that this solution introduces with already existing solutions. To do this, the most relevant Web engineering approaches were studied in detail.

Once the solution to the problem was described, we developed and validated it (steps 3 and 4). These two steps were performed in several phases (see Figure 1.1):

- First, we identified the abstractions that describe Web applications at the requirements level in order to define the Task-based Requirements Model. This model was validated through the development of several cases case studies.
- Second, the RE tool for supporting the creation of task-based requirements models was developed.
- In parallel to this second phase, we identified the correspondences between the task-based requirements model and the OOWS conceptual model. We defined these correspondences as graph transformation rules. We validated these rules by applying them to several case studies by means of the AGG tool. For this validation, models are represented as graphs manually.
- Once (1) graph transformation rules were validated and the viability of the automatic derivation between models has been checked and (2) the format in which the RE tool store task-based requirements models were clearly defined, we developed tools for automatically representing task-based requirements model as graphs, and for obtaining an OOWS conceptual model from a graph that represents it. These tools were validated through several case studies.

Finally, we analyzed the results of our research work in order to obtain several conclusions as well as to delimitate areas for further research (step 5).

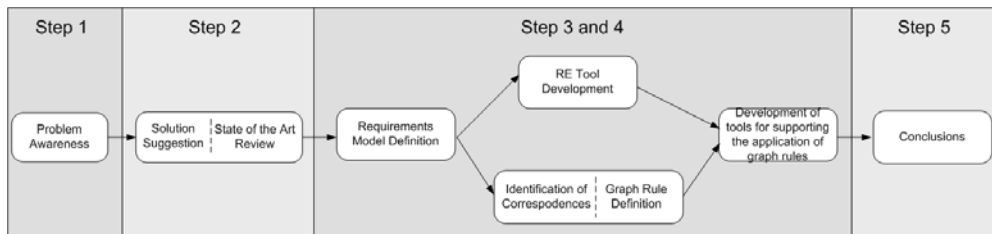


Figure 1.1 Research methodology followed in this thesis

1. Introduction

1.6 The Context of the Thesis

This thesis was developed in the context of the Research Group Object-Oriented Methods for Software Development (OO-Method Group) of the Technical University of Valencia (UPV – Universidad Politécnica de Valencia).

The work presented in this thesis arises from the work performed by a subgroup of researchers of the OO-Method Group. The author has actively participated in both the conception of this work and its development. The work that has made the development of this thesis possible is in the context of the following R&D government projects:

”WEST: WEb-oriented Software Technology” CYTED Project VII-18 (Iberoamerican Program for the Development of Science and Technology). From August 2000 to August 2003.

”Web Environment Engineering (Ingeniería de Ambientes Web)” CICYT Project. Ref. TIC2001-3530-C02-01. UPV and University of Alicante. From 2002 to 2004.

”Desarrollo De E-Servicios Para La Nueva Sociedad Digital (Destino)”. National Project I+D+I 2004-2007, Ref. TIN2004-03534.

1.7 Thesis Structure

The rest of this thesis is organized as follows:

Chapter 2: State of the Art

This chapter presents a critical analysis of the most well-known Web engineering approaches for the development of Web applications. In particular, we focus on how Web application requirements are specified in these approaches. We also study whether or not guidelines for the creation of Web application conceptual models from requirements are proposed.

In this analysis, we present and compare the techniques that Web engineering approaches introduce to specify requirements. We identify the main limitations of these techniques and conclude with a summary of the main drawbacks in Web application requirements specification that should be improved. We take these drawbacks as a base in order to develop the work of this thesis.

Chapter 3: A Web Application Development Process

1. Introduction

In this chapter, we introduce the OOWS development process that has been extended to support RE activities. This process is based on the MDD principles and provides support to different stages of the development of a Web application: requirements, conceptual modelling and implementation. Furthermore, we introduce an iterative and prototyping RE process for improving the requirements stage. The different processes introduced in this chapter are defined by using the SPEM notation [SPEM 2002].

Chapter 4: A Task-Based Requirements Model for Specifying Web Applications

This chapter introduces a requirements model, which is based on the task metaphor, that provides mechanisms for the specification of Web application requirements. The concept of task is reoriented in this model to properly consider the navigational aspect of Web applications. In order to clearly map the new concepts and abstraction mechanisms introduced in this model, an E-commerce application like the Amazon web site has been taken as a case study.

Furthermore, this chapter also presents a RE tool that allows us to visually create the task-based requirements models. This RE tool has been implemented by using a technology based on the Eclipse Platform [Eclipse].

Chapter 5: Requirements Traceability Rules

This chapter introduces a traceability rules catalog, which is another important contribution of this thesis. It represents the set of relevant rules that makes an explicit interpretation of task based requirements models into OOWS conceptual models. Although it is not possible to derive complete OOWS conceptual models from the task-based requirements model, we obtain a conceptual model *skeleton* that constitutes a valuable starting point for the conceptual modelling stage. Furthermore, the traceability rules presented in this chapter constitute a key factor in the generation of Web application prototypes from requirements specifications.

Chapter 6: Applying Traceability Rules through Model-to-Model Transformations

This chapter presents a strategy for automatically interpreting task-based requirements models according to the traceability rules presented in Chapter 5. To do this, traceability rules are defined as graph transformation rules. Next, a tool supported strategy is presented in order to automatically apply these graph transformation rules and then automatically obtain OOWS conceptual models from task-based requirements models.

1. Introduction

Chapter 7: Prototyping Requirements

This chapter presents a general view of how the tools proposed in this thesis are used to generate Web application prototypes from requirements models. We also analyze the different results that are obtained by each tool and how they are taken as source by other tools. In this aspect, we pay special attention to the Web application prototype that is generated and we analyze how it supports the requirements specified in a task-based requirements model.

Chapter 8: Conclusions

This chapter concludes by discussing the appropriateness of the solution proposed in this thesis. Contributions are summarized and future works are proposed. We also introduce the publications that the work of this thesis has originated.

Appendix A: Meta-Model of the Task-based Requirements Model

This appendix presents the meta-model of the task-based requirements models presented in Chapter 4.

Appendix B: The OOWS Approach in a Nut Shell

This appendix summarizes the main aspects of the Web engineering method OOWS.

Appendix C: Some Essentials of Graph Transformations

The technique that is presented in Chapter 6 for performing model-to-model transformation is based on graph transformations. This appendix presents the formalism in which the graph transformation technique is based.

Appendix D: A CASE Study: An Amazon-like E-Commerce Application

In this appendix, we apply the approach presented in this thesis to the running example used throughout the thesis. This running example is an E-commerce Application like Amazon. We present the task-based requirements model that specifies the requirements of this case study. Furthermore, we show the results obtained after applying the different steps proposed in the model-to-model transformation strategy.

Chapter 2

*“I have never met a man so ignorant that
I couldn't learn something from him.”*

Galileo Galilei.
(Italian Physicist, Mathematic, Astronomer,
and Philosopher, 1564-1642)

2 State of the Art

In this chapter, we present an analysis of the most important Web engineering approaches that have been proposed to support the development of Web applications. In particular, we analyze first how these approaches face the specification of Web application requirements as well as whether or not mechanisms to guide analysts in the creation of conceptual models from requirements are provided. Next, we discuss the main limitations that currently exist in the specification of Web application requirements by taking into account the analysis introduced previously.

Before presenting this analysis some background about requirements specification is introduced.

2.1 Requirements Specification in the context of Web engineering

Requirements specification is the activity by means of which the requirements that a software system must satisfy are described in a catalogue.

2. State of the Art

According to the IEEE Standard-830-1998 [IEEE 1998], a requirement is: **(A)** A condition or capability needed by a user to solve a problem or achieve an objective. **(B)** A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document. **(C)** A documented representation of a condition or capability as in definition (A) or (B).

Requirements can be classified in different types depending on the capabilities that they support. In the context of Web applications, a requirements classification is proposed in [Escalona & Koch. 2004]. According to this classification there are two main types of requirements:

- **Functional requirements** that are capabilities that a system must possess in order to solve a problem. They can be sub-classified into the following types:
 - *Data requirements*, also known as conceptual requirements, content requirements or storage requirements. These requirements establish how information is stored and administrated by the Web application.
 - *Interface requirements* (to the user) also known as interaction requirements or user's requirements. They give an answer to how the user is going to interact with the Web application.
 - *Navigational requirements* represent users' navigation needs through the hyperspace.
 - *Personalization requirements* also known as customization or adaptation requirements. They describe how a Web application has to (dynamically) adapt itself, depending on the user or environment profile.
 - *Transactional requirements*, also known as internal functional requirements or service requirements, express what the Web application has to compute internally, without considering interface and interaction aspects.
- **Non-functional requirements** act to constraint the solution, e.g. portability requirements; reuse requirements, usability requirements, availability requirements, performance requirements, etc.

The approach presented in this thesis is focused on specifying the structural, behavioural and navigational aspects of Web applications at the requirements level. Thus, we introduce next the techniques that most relevant Web engineering methods use to specify the following types of requirements: data and transaction requirements (which capture the structural and the behavioural aspects) and navigational requirements (which capture the navigational aspect).

2.2 Study of Web Engineering Methods

In this section, we study the most relevant methods in Web engineering, paying special attention to how Web application requirements are specified. In this context, we only present in detail those Web engineering methods that introduce a requirements phase in its development process (from Section 2.1.1 to Section 2.1.9). They are presented chronologically according to the year in which they appeared. For each of these methods we present the following information:

- A general description of the method.
- The phases of its development process
- A study of the requirements specification techniques that are proposed by the method. To present a representative example of the requirements specifications obtained by these techniques we use them to specify the requirements of an E-commerce application like Amazon⁵ (hereafter known as the Amazon example).
- The main limitations of these techniques.
- Tool support. In particular, we focus on analyzing whether or not tools for supporting the specification of Web application requirements are provided.
- The guidelines and/or tools that are provided to obtain the Web application conceptual model from requirements specifications.

We have studied other approaches that are not considered in this detailed analysis because they do not provide explicit support for the specification of Web application requirements. These approaches are introduced in a schematic way in Section 2.1.10 and Section 2.1.11. Section 2.1.10 introduces those approaches that only focus on providing new development processes for Web applications. These approaches only indicate which activities must be performed in order to develop Web applications. They do not propose new techniques for supporting these activities. Section 2.1.11 introduces those approaches that support the development of Web application but do not consider requirements in its development process. These techniques are mainly focused on providing models for describing Web applications in a high level of abstraction.

2.2.1 OOHDM: Object Oriented Hypermedia Design Model

OOHDM was developed by Schwabe and Rossi in 1994 [Schwabe & Rossi 1994] [Schwabe *et al.* 1996]. It was one of the first approaches in providing a methodological solution for the development of Web applications. This approach takes some ideas proposed in HDM [Garzotto *et al.* 1993] and applies them in a well-defined development process based on the Object-Oriented paradigm. OOHDM

⁵ <http://www.amazon.com>

2. State of the Art

emphasizes the separation of the navigational aspect from other aspect such as the conceptual aspect and the interface aspect. Other approaches have been further inspired by this idea of separation of aspects. Finally, it is worth to remark that OOHDM is not a closed approach and it is continuously being extended and improved.

Development Process. The development process of this approach is divided into five main phases:

- *Requirements Gathering:* In this phase, the users that must interact with the Web application are identified as well as the users' needs that the Web application must support. Early versions of OOHDM do not consider the requirements phase. This phase was included after defining some extensions to the method [Vilain et al. 2000b].
- *Conceptual Design:* This phase consists in the definition of a conceptual schema where the static aspect of the system is described.
- *Navigational Design:* In this phase, we define a navigation class diagram and a navigation structure diagram. The first one represents the static possibilities of navigation in the system. The second one extends the navigation class diagram including access structures and navigation contexts.
- *Abstract Interface Design:* This phase consists in the description of the user interface in an abstract way. To do this, an abstract interface model is developed using a special technique named ADVs [Cowan & Lucena 1995].
- *Implementation:* In this phase, the Web application is implemented. It is based on the previous models.

Requirements Specification Techniques: Web application requirements are handled in the phase of Requirements Gathering. This phase is divided into five steps:

1. Identification of Roles
2. Specification of Scenarios
3. Specification of Use Cases
4. Specification of User Interaction Diagrams
5. Validation of Use Cases and User Interaction Diagrams

In this context, requirements are specified as follow: (1) Once user roles are identified, (2) users describe the work that each role must perform by means of scenarios; next, (3) use cases are defined from scenarios and finally (4) use cases are refined with user interaction diagrams (UIDs). For each use case, a UID is defined. Each UID graphically describes the interaction between the users and the system without considering specific aspects of the user interface. The process to get an UID from a use case is described very carefully in the approach. Figure 2.1 shows a partial requirements specification of the Amazon example that has been defined by using the OOHDM approach.

2. State of the Art

Figure 2.1 shows three scenarios described by different users: Browsing CDs, Finding a CD and Adding a CD to the shopping cart. These three scenarios describe a similar work. Then, a use case that supports this work is derived from them. The UID associated to this use case indicates that: the system provides users first with a list of music category names. From this list, users can select one and then they obtain a list of CDs. The title, the price and the artist name is given for each CD. From the list of music categories, users can also introduce the name of an artist. From this name, a list of artists is provided. If users select one artist they obtain a list with its CDs. From the list of CDs, users can select one and then obtain a description of it. In this description, the title, the price, the songs, the cover, some comments and the artist name of the CD are provided. Users can also add the CD to the shopping cart at this point.

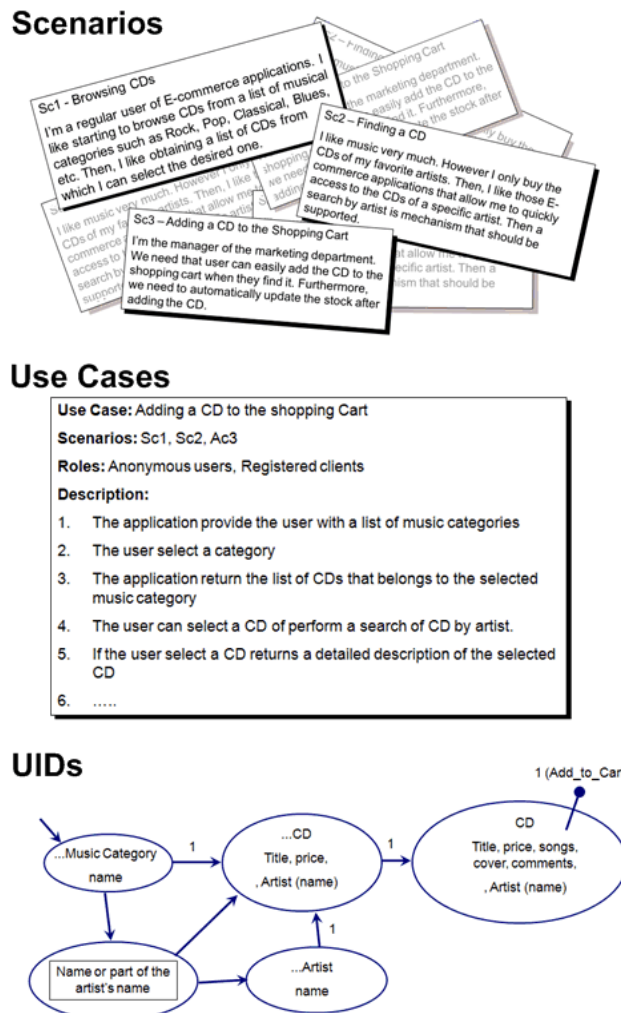


Figure 2.1 Example of requirements specification in OOHDM

2. State of the Art

Requirements Specification Limitations: Navigation is captured in a narrow way. Navigation is captured for each use case individually (from its associated UID) and then navigation is not captured from a global view of the system. In this context, navigational requirements captured by UID are not properly related to each other (e.g. how users access different navigational structures captured by different UIDs). Furthermore, although transactional requirements can be textually described in a use case they cannot be completely specified in UIDs. For instance, a sequence of system operations cannot be defined in a UID. On the other side, data requirements must be extracted from interactions between the user and the system which is not always adequate. For instance, data that is only required to support internal operations of the system is not properly considered.

Tool Support: Currently, there is no tool which supports the Requirements Gathering phase. Then, each step must be manually performed.

Guidelines for Obtaining Conceptual Models: This approach presents guidelines in order to help analysts to define the conceptual and navigational schema from requirements specifications. These guidelines are defined from a set of rules presented in [Vilain et al. 2000b] and [Vilian & Schwabe 2002]. These rules indicate how the elements that define both a class diagram (conceptual schema) and a context diagram (navigational schema) can be systematically derived from UIDs.

These rules are textually described by using natural language. No formalism is used in the rule definition. In this context, certain ambiguity degree is introduced in the rule definition (derived from the use of natural language). This aspect becomes critical if we consider that rules must be manually interpreted and applied. No tool is provided to support the application of these rules.

2.2.2 WSDM: Web Site Design Method

WSDM was developed by De Troyer and Leune in 1998 [De Troyer & Leune 1998]. It is a user-centred approach. WSDM defines a Web application by describing the requirements of the different groups of users that must interact with it. It was one of the first approaches in considering the problem of the diversity of users in Web applications. This approach is continuously being extended and improved.

Development Process: The development process of this approach is divided into five main phases:

- *Mission Statement Specification:* In this phase, we must express the purpose and the subject of the Web application. We must also declare the target audience.
- *User Modelling:* In this phase, users are classified and grouped in order to study system requirements according to each user group.

2. State of the Art

- *Conceptual Design*: In this phase, both a class diagram is designed to represent the static model of the system, and a navigational model to represent the possibilities of navigation.
- *Implementation Design*: This phase consists in the translation of the models defined in the conceptual design phase into an abstract language easily to be understood by the computer.
- *Implementation*: In this phase, the implementation design result is written in a specific programming language.

Requirements Specification Techniques: Requirements are handled in the Mission Statement Specification and User Modelling phases. Taking as source the description of the mission statement, the User Modelling phase is performed in two steps: Audience Classification and Audience Class Characterization. In these two steps the concept of class audience plays a key role. A Class Audience is a group of potential visitors that has the same functional requirements. Then:

1. In the Audience Classification step, people are classified according to the activities related to the purpose and subject of the Web application. This formally identifies the different Audience Classes. WSDM proposes to analyze the organization environment where the application will be used, and centers the attention on the stakeholders of the business processes supported by the application. In WSDM the relationships between stakeholders and the business process activities performed are graphically represented by conceptual maps of roles and activities.
2. In the Audience Class Characterization step, Audience Classes are analyzed in detail. To do this, textual templates with the help of a data dictionary are used to define the functional requirements for each group of users.

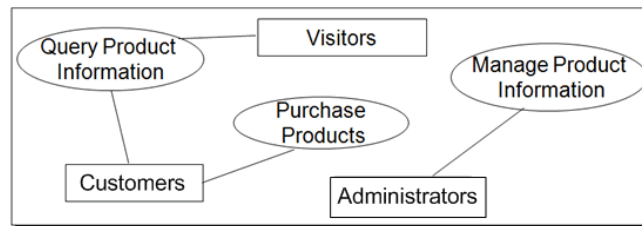
Figure 2.2 shows a partial requirements specification of the Amazon example that has been defined by using the WSDM approach. The upper side of this figure shows the mission statement. Below the mission statement we can see a partial result of the audience classification step. Three audience classes have been identified: Customers, Visitors and Administrators. Customers can query product information and purchase products; Visitors can only query product information; Administrators can manage product information. Finally, in the lower side of Figure 2.2 we can see a partial characterization of the audience class Visitors.

2. State of the Art

Mission Statement

To support users (Visitors, Customers, and Administrators) in (1) the purchase of CDs, Books and Software products and (2) the management of the system throughout the Internet.

Audience Classification



Audience Characterization

Visitors

1. View the different type of products. For each type view the list of available products.
2. View product descriptions. For each product, get a description. Link the product's name in the list of products to a product description web page.
3. Search products. For every type, users can search products by introducing a specific criterion.
4. ...

Figure 2.2 Example of requirements specification in WSDM

Requirements Specification Limitations: The main drawback of this approach is that requirements are basically defined textually. In this context, describing navigational requirements by textual descriptions (which may need to consider multiples possibilities of navigation information) is not always an easy task. These textual descriptions may be overloaded and difficult to manage in the development of complex Web applications. Furthermore, as works such as [Vilian et al. 2000a] state, textual specifications are unwieldy when are used in a validation process with users, mainly because of the lack of precision and conciseness.

Tool Support: Currently, there is no tool for supporting the requirement phases. Then, requirements must be manually specified.

2. State of the Art

Guidelines for Obtaining Conceptual Models: There are no published works that present guidelines that help analysts to define the Web application conceptual model from the requirements specification.

2.2.3 SOHDM: Scenario-based Object-Oriented Hypermedia Design Methodology

SOHDM was developed by Heeseok Lee, Choongseok Lee, and Cheonsoo Yoo in 1998 [Lee *et al.* 1998]. This approach considered RE activities in the development process of a Web application from its initial version. This approach takes some ideas from OOHDM [Schwabe *et al.* 1996]. However, as its name indicates it is mainly based on the use of scenarios.

Development Process: The development process of this approach is divided into six main phases:

- *Analysis:* In this phase, the requirements of the Web Application are describe by using scenarios.
- *Object model realization:* This phase consists in creating a class diagram where the static structure of the system is defined.
- *View design:* In this phase, we express how the system will be presented to the user.
- *Navigational design:* In this phase, a navigational class model is developed in order to express the possibilities of navigation in the system.
- *Realization of the implementation:* This phase consists in designing the Web pages and the flow among them, other detailed interface aspects and a relational database.
- *Construction of the system:* In this phase the Web application is finally built.

Requirements Specification Techniques: Requirements are handled in the Analysis phase. SOHDM proposes to specify Web applicaiton requirements through three main steps:

1. Definition of the scope of the system, which delimits the Web application to be developed from external entities. In order to define the scope of the system SOHDM proposes the System Scope diagram that is based on the well-known Data Flow Diagrams (DFD) [DeMarco 1979], although context diagrams (e.g., zero level DFD) are a good alternative. In the System Scope diagram, we identify external entities as well as the information exchanged between these external entities and the system.

2. State of the Art

2. Identification of events that trigger the communication between external entities and the application. For each external entity one or more events are identified. Events are defined in a specific table.
3. Association of scenarios to the identified events. Scenarios are described by means of Scenario Activity Charts (SACs) that is a notation created by the authors. SAC describes business processes according to actors. An actor is an operator of specific activities, i.e., a creator of events.

Figure 2.3 shows a partial requirements specification of the Amazon example that has been defined by using the SOHDM approach. The upper side of this figure shows the scope of the Web application where three external entities are identified: Visitor, Customer and Administrator. Below the system scope, we can found the events identified for each external entity. In the lower side of this figure, the SAC that describes the event Query Product Information is shown. External entities are the primary candidates for actors in SACs. An event is a starting point, a trigger of a scenario. An activity is an operation by which an actor completes a scenario. A decision node represents an activity that enables for an actor to choose the next activity. An activity flow is a sequence of activities. Termination is the end of a scenario.

Requirements Specification Limitations: The main drawback of this approach is that SACs are mainly focused on describing transactional requirements where the exchange of information between the system and the user is almost limited to required operation parameters. Thus, the description of navigational and data requirements is not always easy in the case of Web applications that are mainly focused on just providing information. The major part of these Web applications allows users to navigate information (data and navigational requirements) without performing any operation (transactional requirements). In this context, to represent all the different possibilities for navigating information with a DFD-based notation can overload the requirements specification, making them difficult to manage and understand. Furthermore, this technique does not provide explicit support for the specification of data requirements. Only the data that is requested by the system in order to perform an activity can be defined in a SAC.

Tool Support: Currently, there is no tool for supporting the Analysis phase. Then, every diagram must be manually defined.

Guidelines for Obtaining Conceptual Models: There are no published works that present guidelines that help analysts to define the Web application conceptual model from the requirements specification.

2. State of the Art

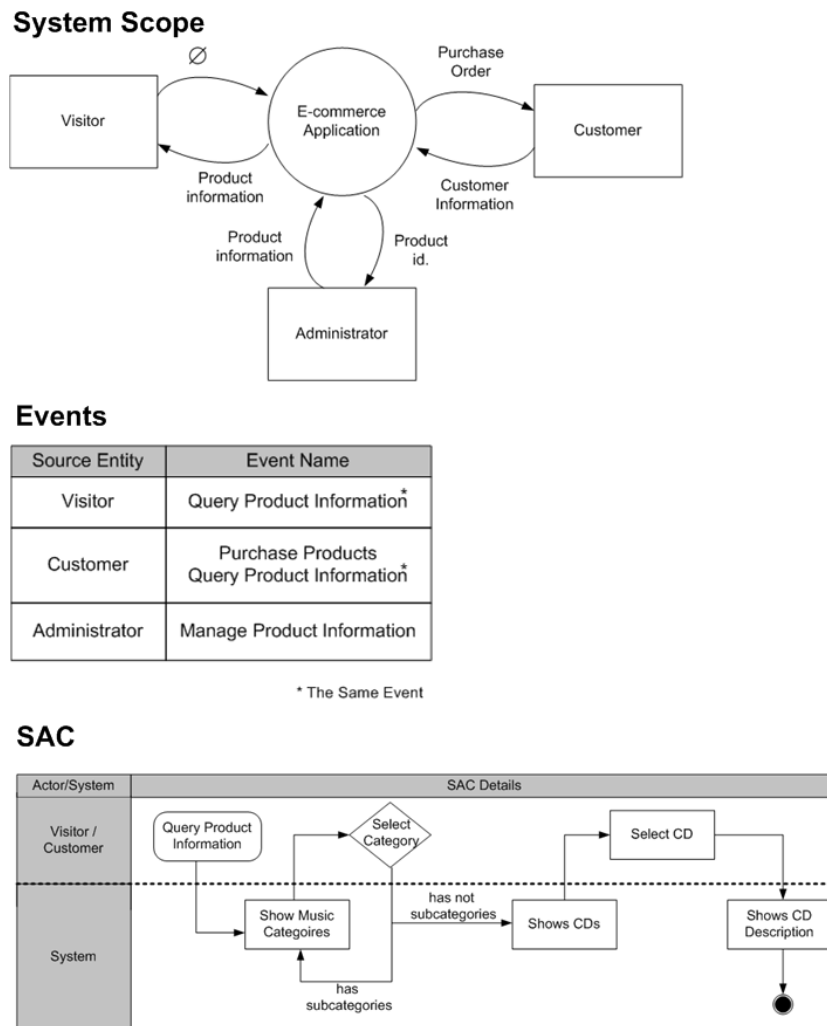


Figure 2.3 Example of requirements specification in SOHDM

2.2.4 UWE: UML-based Web Engineering

UWE has been developed by Nora Koch from 1998 [Mandel *et al.* 1998] [Koch 2002] [Koch et al. 2006]. UWE is a methodological approach for the development of Web applications that completely bases its diagrammatic techniques on UML. This approach is currently open and a lot of research work is continuously being presented in order to improve it. UWE proposes a semi-automatic design process supported by the case tool ArgoUWE⁶.

⁶ <http://www.pst.informatik.uni-muenchen.de/projekte/uwe/home.shtml>

2. State of the Art

Development Process: The development process of UWE is based on the Model Driven Architecture [MDA]. The aim of such a MDD process is the automatic model transformation in each step based on rules defined at meta-model level. We can divide this process in three main stages:

- The process starts with the specification of requirements. These requirements define the CIM model.
- Platform independent analysis models (PIMs) are derived from the requirements (often based on additional information). On the PIM level, the separate concerns of Web applications (the content, the navigation, the business processes, and the presentation) are designed in separate models. These models are integrated into a so-called “big picture” which merges all the concerns together and is used in validation of the design models.
- Finally, the platform specific models (PSMs) are derived from this validation model, from which program code can be generated.

Requirements Specification Techniques: Initial versions of UWE propose the specification of requirements by means of use case diagrams together with textual descriptions of use cases [Koch 2001]. In recent works [Koch et al. 2006]⁷, UWE has been extended in order to support the UML profile for Web requirements (WebRE) [Escalona et al. 2006]⁸. This profile has been defined by the UWE’s authors in collaboration with the authors of the Web engineering method NDT (further presented). We focus on the most recent approach based on WebRE, considering that the other works could be obsolete.

In this context, UWE currently proposes the use of use cases diagrams and activity diagrams in order to capture Web application requirements. Use case diagrams are used to represent an overview of the functional requirements while activity diagrams provide a more detailed view. UWE distinguishes between two types of use cases: *Navigation use cases* (marked with □) which comprise a set of browse activities (marked with ⇒) that the user will perform to reach a target node. A browse activity is the action of following a link. A browse activity can be enriched by search actions (marked with ?). A search action has a set of parameters, which let define queries on a content (marked with ○). The results are shown in the target node (marked with □). The second kind of use case is the *WebProcess* (marked with ⌢), which is refined by activities of type browse, search, and at least one user transaction (marked with ⇔). A user transaction represents more complex activities that are expressed in terms of transactions that have been initiated by the user, like checkout in an e-shop or an online reservation.

^{7,8} It is worth to remark that these works are later to initial published works of this thesis such as [Valderas et al. 2005a] or [Valderas et al. 2005b]

2. State of the Art

Figure 2.4 shows a partial requirements specification of the Amazon example that has been defined by using the UWE approach. The upper side of this figure shows the use case diagram. The lower side shows the activity diagram that describes the navigational use case Find CD. According to this diagram, when users activate the link List Music Category they access the node Music Category. From this list users can select one category and then access the node CD. Another way of accessing this node is by searching artists from the node Music Category.

Requirements Specification Limitations: UWE together with OOHDM are two of the few Web Engineering approaches that provide specific techniques for the specification of Web application requirements. In fact, both approaches share some ideas. For instance, both use a use case diagram and then complement uses cases with a more detailed description in order to describe the requirements of each use case.

In this context, some of the problems that we have detected for OOHDM can also be applied to UWE. In particular, they consider navigational requirements from a very narrow way, considering only those requirements which support each use case individually. UWE does not analyze navigational requirements from a global vision of the system. Furthermore, data requirements must be extracted either from the parameters that are needed for performing searches or from the information that is modified by transactions performed by the user [Koch et al. 2006]. As we have already explained in the analysis of OOHDM, this aspect makes it difficult to consider the requirements related to data that is only required to support internal operations of the system. Finally, activity diagrams in UWE allow us to indicate which type of information is accessed by users by navigating nodes (e.g. CDs, Music Categories, etc). However, it is not clear how details about the information provided in each node can be given at the requirements level.

Tool Support: UWE is supported by the ArgoUWE tool. This tool is a plugin defined over the open source modelling tool ArgoUML⁹. ArgoUWE is focused on supporting the modelling activities of the UWE development process as well as activities related to the semi-automatic generation of code from models. In this context, both use case diagrams and activity diagrams can be graphically defined by means of ArgoUWE. Additional information about ArgoUWE can be found in its Web page¹⁰.

Guidelines for Obtaining Conceptual Models: A model-to-model transformation is proposed in [Koch et al. 2006]¹¹ in order to obtain draft versions of both UWE navigational models and UWE content models from requirements specifications. These model-to-model transformations are based on the definition of mappings

⁹ <http://www.argouml.org>

¹⁰ <http://www.pst.informatik.uni-muenchen.de/projekte/uwe/home.shtml>

¹¹ This work is also later to initial published works of this thesis such as [Valderas et al. 2005b]

2. State of the Art

between the WebRe meta-model and the UWE meta-model. These mappings are defined as QVT transformations [QVT]. However, authors do not provide a tool for automatically applying these transformations. They are currently looking forward to tools supporting the QVT transformation language. In the meantime, they are gathering experience with other transformation languages and techniques, such as ATL and graph transformations [Segura *et al.* 2007]¹².

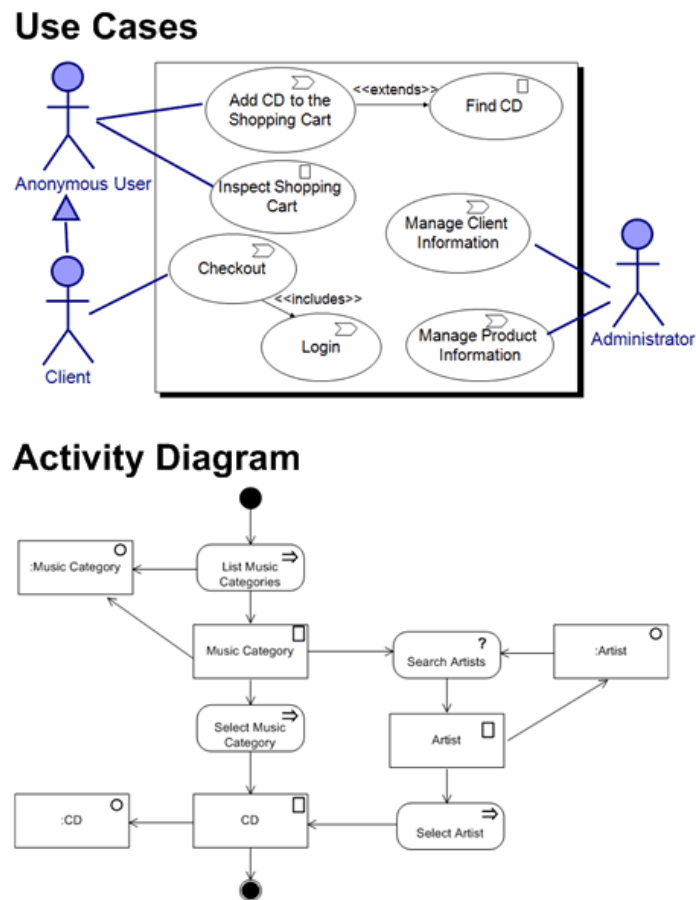


Figure 2.4 Example of requirements specification in UWE

2.2.5 Building Web Applications with UML

This approach is an extension to UML introduced by Jim Conallen in 1999 [Conallen 1999]. It is based on the Unified Process [Jacobson *et al.* 1999]. The graphical

¹²This work is also later to initial published works of this thesis such as [Valderas *et al.* 2005b]

2. State of the Art

notation of UML is extended throughout stereotypes. These stereotypes can be freely obtained and incorporated to the IBM tool Rational Rose¹³.

Development Process: The development process of this approach is divided into eight main phases:

- *Planning*: This phase consists in the analysis of the organization needs and the definition of a plan of how the rest of the phases in the development process must be performed.
- *Requirements*: In this phase, the requirements of the system are gathered and defined by means of UML-based models such as use case diagrams or activity diagrams.
- *Analysis - Design*: These two phases consist in the transformation of requirements into a design that can be realized in software. These two phases can be done separately or combined as a part of one set of activities.
 - Analysis consists in obtaining an analysis model from requirements. This model is made up of classes and collaboration of classes that represent the dynamic behaviour of the system. The emphasis of this phase is on ensuring that all the functional requirements are supported.
 - Design consists in the use of non-functional requirements and architecture constraints in order to refine the analysis model into something that can be coded.
- *Implementation*: In this phase, the Web application design is mapped into code and components. A set of mappings are provided to facilitate this. Next, code must be build into binaries.
- *Test*: In this phase, each member of the development team must test its own work. The goal of this phase is to have every delivered artefact, or component, tested before any other member of the team uses it.
- *Evaluation*: The whole system is evaluated to ensure that correctly supports users' needs.
- *Deployment and Maintenance*: In this phase, the system is delivered and installed. After that, maintenance activities must be performed.

Requirements Specification Techniques: Requirements are handled in the Requirements phase. Although this approach proposes multiple new UML stereotypes for supporting the Web application development they are all mainly focused on the analysis and design phase. The technique proposed to specify requirements is based on the use of traditional use cases together with activity diagrams and/or sequence diagrams. Activity diagrams are proposed to describe use cases. Sequence diagrams are recommended to be defined in order to better clarify which is the basic flow of the use case and which are the alternative flows.

¹³ <http://www.ibm.com/software/awdtools/developer/rose/index.html>

2. State of the Art

Figure 2.5 shows a partial view of the Conallen's requirements specification for the Amazon example. The upper side of this figure shows the use case diagram. The lower side shows the activity diagram associated to the use case Find CD.

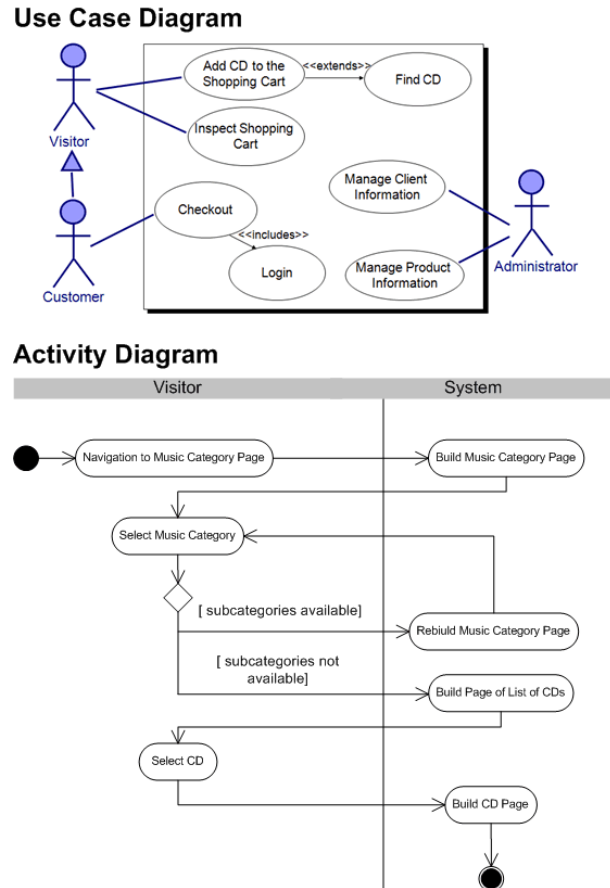


Figure 2.5 Example of requirements specification in the Conallen's approach

Requirements Specification Limitations: This approach does not provide specific techniques for handling Web application requirements and it proposes the use of traditional use cases. Use cases become some times insufficient and too ambiguous when they are used to capture the navigational aspect of Web applications [Insfran *et al.* 2002] and [Vilain *et al.* 2000b]. Although in the Conallen's approach a diagrammatic technique such as Activity Diagrams is proposed to complement use cases, it is based on concepts that are close to the implementation level. Notice how in order to describe the use case Find CD we need to indicate the web pages that must support the Web application. In this thesis, we provide a more abstract technique in order to take such decisions in later phases of the development process. Furthermore, the technique proposed by Conallen is mainly focused on capturing the behavior of

2. State of the Art

both the user and the system. Few support for detailing structural aspects such as the data that the system must store are provided.

Tool Support: Since this approach is fully based on UML it is supported by general modelling tools such as Rational Rose. In fact, profiles proposed by this approach are available to be incorporated in this tool.

Guidelines for Obtaining Conceptual Models: There are no published works that present guidelines to facilitate the creation of Web application conceptual models from requirements specifications.

2.2.6 WebML: Web Modelling Language

WebML was developed by Piero Fraternali and Stephano Ceri in 2000 [Ceri et al. 2000]. WebML is a high level modelling language for Web applications. WebML follows the style of both Entity-Relationship and UML offering a proprietary notation and a graphical representation using the UML syntax. This approach is currently not closed and it is continuously being extended and improved. Finally, WebML is one of the few approaches that presents a tool that supports its development process. This tool is called WebRatio¹⁴ and it is being currently applied in an industrial environment.

Development Process: The development process of this approach is divided into six incremental main phases:

- *Requirements Analysis:* This phase focuses on collecting information about the application domain and the expected functions, and specifying them.
- *Conceptual Modelling:* This phase consists in defining WebML-based conceptual schemas, which express the organization of the application at a high level of abstraction, independently from implementation details.
- *Implementation:* This phase consists in the translation of the WebML-based conceptual schema into a specific implementation technology.
- *Testing and Evaluation:* In this phase, the Web application is tested and validated in order to improve its internal and external quality.
- *Deployment:* This phase consists in deploying the Web application on the top of a specific architecture.
- *Maintenance and Evolution:* In this phase, the application is maintained and possibly evolved after it is deployed.

Furthermore WebML also promotes a strategy of rapid prototyping from the conceptual schema in order to early detect misunderstanding in the captured requirements [Fraternali et al. 2006] [Ceri et al 2001].

¹⁴ [Http://www.webratio.com](http://www.webratio.com)

2. State of the Art

Requirements Specification Techniques: Although WebML considers RE activities in its development process it does not prescribe any specific format for requirements specification. According to its authors [Brambilla *et al.* 2006]: “Requirements specification is the activity in which the application analyst collects and formalizes the essential information about the application domain and expected functions. This aspect does not significantly differ from requirement collection for traditional applications.”

Although no specific techniques for Web application requirements specification are given, we can see along the different published works that are related to WebML how authors suggest some tabular formats for capturing types of users [Matera *et al.* 2003] or propose the use of UML uses cases and activity diagrams in order to specifying functional requirements [Ceri *et al.* 2001]. Furthermore, the use of mock-ups (sketches) is also suggested for the specification of the site view [Ceri *et al.* 2006].

Figure 2.6 shows a partial view of the requirements specification of the Amazon example that has been defined by using the techniques proposed by WebML. In particular, the upper side of this figure shows the description of the Visitor user type. Below this description, we find an activity diagram that group in phases the different activities that can be performed by the identified types of user. In the lower side of this figure, we can find a use case diagram that provides a more detailed description of one of the phases included in the activity diagram (Collect Products).

Requirements Specification Limitations: The main drawback of this approach is that it is focused on the use of traditional techniques such as use cases without an explicit extension for supporting Web application requirements. As we have explained in the previous approach, navigational requirements are many times not supported properly by use cases [Insfran *et al.* 2002] and [Vilain *et al.* 2000b]. In the same way, other works such as England & Finney (1999), Burdman (1999), Overmyer (2000) and Lowe (2003) also state that new RE methods are needed for Web applications because of RE methods for traditional software are not appropriate to support distinctive characteristics of the Web application development such as, for instance, navigation.

Tool Support: As introduced above, WebML is supported by the commercial tool WebRatio. However, this tool is mainly focused on (1) providing support for the Conceptual Modelling phase and (2) providing mechanisms for the automatic generation of code. The specification of requirements is not supported by this tool (at least in its current version 5.0).

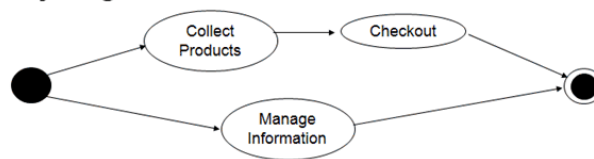
Guidelines for Obtaining Conceptual Models: There are no published works that present guidelines to define the Web application conceptual model from the requirements specification.

2. State of the Art

User Type Description

Visitor
Description: It represents all the users that can access the public part of the Web application.
Profile data: No profile required
Object access in read mode: Products
Object access in write mode: Shopping Cart
Relevant usage scenarios: Query Product Information, Add product to shopping cart

Activity Diagram



Use Case Diagram

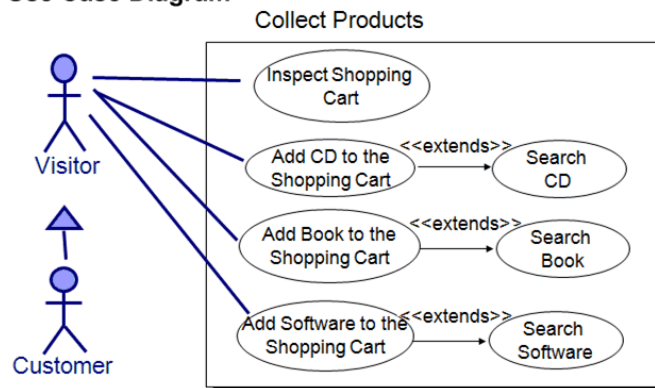


Figure 2.6 Example of requirements specification in WebML

2.2.7 OO-H: Object-Oriented Hypermedia Method

Object-Oriented Hypermedia Method was presented by Jaime Gomez, Cristina Cachero and Oscar Pastor in 2000 [Gomez *et al.* 2000] [Cachero 2003]. This approach emerged formerly as an extension of OO-Method [Pastor *et al.* 1997], a design method for object-oriented systems. However, in the most recent publications authors dissociate OOH from OO-Method and focus on dealing with personalization of web sites [Garrigos *et al.* 2005]. This approach is supported by the tool Visual Wade¹⁵.

¹⁵ <http://www.visualwade.com/>

2. State of the Art

Development process: The development process of this approach is divided into four main stages:

- *Requirements Analysis:* In this phase, the Web application requirements are specified for each different type of user.
- *Engineering:* This phase involves all the activities related to the analysis and design of the software product. In particular, these activities are five: two analysis activities (domain and navigational analysis) and three design activities (domain, navigation and presentation design).
- *Construction and Adaptation:* This phase constitutes the implementation of the Web application.
- *Client Evaluation:* In this phase, clients evaluate the developed Web application.

Requirements Specification Techniques: Requirements are handled in the Requirements Analysis phase. However, OO-H is mainly focused on both providing abstract mechanisms in order to capture Web applications at the conceptual level and defining strategies for the automatic generation of code. RE activities for Web applications are considered by paying less attention. OO-H proposes only the use of use case diagrams.

In an extension of the method [Cachero and Koch 2002b], OO-H proposes to complement use cases with activity diagrams. However, these activity diagrams are used in the analysis activities that are included in the Engineering phase. They are not used as technique for the specification of Web application requirements.

Figure 2.7 a partial view of the use case diagram proposed by OO-H in order to specify the requirements of the Amazon example.

Requirements Specification Limitations: As happens in other works such as WebML, the main drawback of this approach is the prescription of traditional techniques such as use cases for the specification of Web application requirements.

Tool Support: OO-H is supported by the tool Visual Wade. This tool provides a complete graphical support for performing the domain and navigational analysis. However, this tool (at least in its current version 1.2.163) does not provide support for creating requirements specifications.

Guidelines for Obtaining Conceptual Models: There are no published works that present guidelines to define the Web application conceptual model from the requirements specification.

2. State of the Art

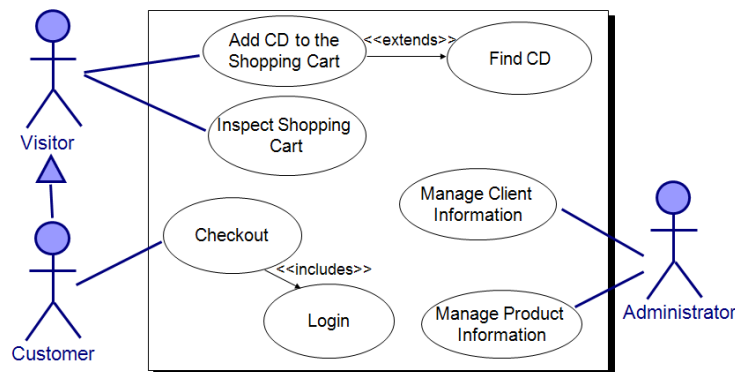


Figure 2.7 Example of requirements specification in OOH

2.2.8 W2000

W2000 was developed by Luciano Baresi, Franca Garzotto, and Paolo Paolini in 2001 [Baresi et al 2001]. This method is presented as an extension and customization of UML with web design concepts borrowed from the Hypermedia Design Model (HDM) [Garzotto et al. 1993].

Development Process: The development process of this approach is divided into five main phases:

- *Requirements Analysis:* This phase consists in the study of the Web application requirements by means of UML use cases.
- *State Evolution Design.* In this phase, analysts must indicate how contents evolve. This phase is not mandatory, but it is required only for applications with complex behaviors.
- *Hypermedia Design.* This phase consists in the design of the Web application navigational structure. This phase is based on the use of HDM.
- *Functional Design.* This phase specifies the main user operations of the application.
- *Visibility design.* In this phase, different perspectives of the Web application are defined for each different type of user.

Requirements Specification Techniques: Requirements are handled in the Requirements Analysis phase. The requirement analysis proposed by W2000 is performed from two main activities: functional requirements analysis and navigational requirements analysis. After identifying the different types of user that can interact with the Web application, the functional requirements analysis identifies the main user operations while the navigational requirements analysis indicates the main information and navigation structures needed by the different users.

2. State of the Art

In order to perform both activities W2000 proposes the use of UML use case diagrams. Two types of use cases are proposed: functional use cases and navigational use cases. The functional use case diagram is a typical use case diagram that identifies the main functionalities and associates them with types of user. The navigational use case diagram indicates the navigation capabilities associated to each type of user. These navigational capabilities can be constrained with accessibility conditions.

Figure 2.8 shows a partial version of the W2000 use case diagrams for the Amazon example. The upper side of this figure shows a functional use case diagram that identifies functionalities such as login or management of product information. These functionalities are associated to the Customer and Administrator types of user, respectively. The lower side of this figure shows a navigational use case diagram that identifies navigational capabilities such as browse CDs or inspects shopping cart items. Notice how this last capability presents an accessibility constraint. Visitors only can inspect the items that they have added to the shopping cart.

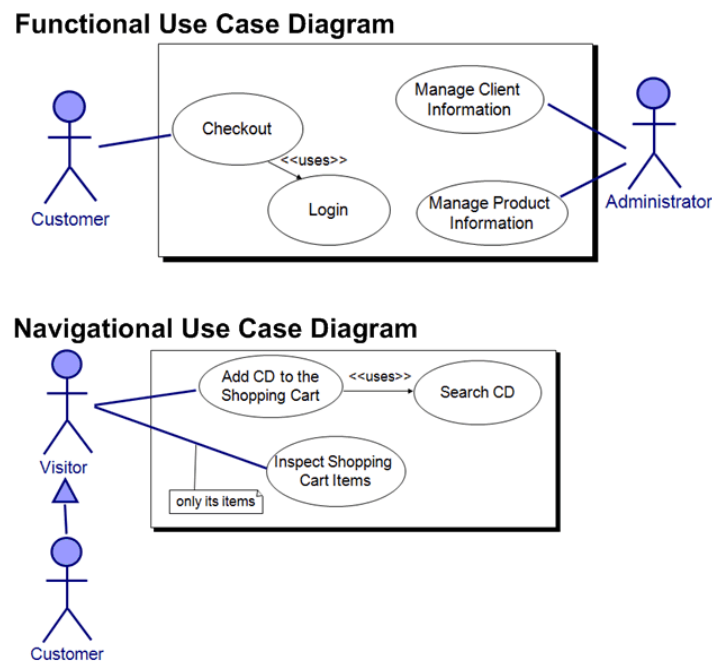


Figure 2.8 Example of requirements specification in W2000

Requirements Specification Limitations: Although W2000 proposes a special use case diagram for capturing navigational requirements, it only allows us to identify navigational capabilities. Techniques for describing at the requirements level how these capabilities must be supported by the Web application (such as the UIDs of OOHDM or the activity diagrams of UWE) are not provided.

2. State of the Art

Tool Support: Currently, there is no tool which supports the RE activities in this approach.

Guidelines for Obtaining Conceptual Models: There are no published works that present guidelines to facilitate the creation of Web application conceptual models.

2.2.9 NDT: Navigational Development Techniques

NDT was presented by Maria Jose Escalona in 2004 [Escalona 2004]. NDT is a methodological process for the Web application development that it is focused on the requirements and analysis phases. It proposes the intensive use of textual templates in the requirements phase and the systematic derivation of analysis models from these templates. This approach proposes the use of prototypes to validate requirements. The NDT tool [Escalona *et al.* 2003] has been developed in order to support this approach.

Development process: The development process of this approach is divided in three main stages:

- *Requirements Treatment:* In this phase, the Web application requirements are collected and described.
- *Analysis:* In this phase, analysis models are systematically derived from the requirements specification. These analysis models are the conceptual model and the navigational model.
- *Prototyping:* This phase consists in the development of Web application prototypes from analysis models. These prototypes are used to validate requirements.

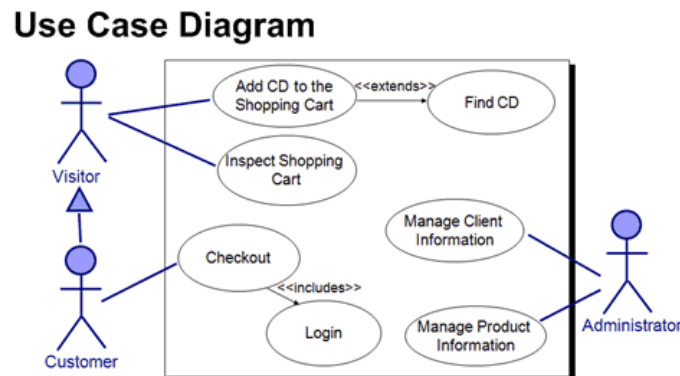
It is worth to remark that the main goal of NDT is to obtain three results: (1) the requirements catalogue, (2) the requirements analysis document and (3) the Web application prototype. These results must be taken as source for the design and implementation phases, which are not handled by NDT.

Requirements Specification Techniques: NDT is an approach fully developed to handle Web applications requirements. Thus, this is one of the approaches that provide a more complete support for the specification of Web application requirements. This approach also fits (together with UWE) the UML profile for Web requirements (WebRE) [Escalona *et al.* 2006].

In order to specify requirements, NDT basically proposes the use of uses cases diagrams and formatted templates. NDT classifies requirements into the following types: storage information, actor, functional, interaction and non-functional requirements. For each type, NDT defines a special template, i.e. a table with specific textual fields that are completed by the development team during the phase of requirements elicitation.

2. State of the Art

Figure 2.9 shows a partial requirements specification of the Amazon example that has been defined by using the NDT approach. The upper side of this figure shows the use case diagram. The lower side shows an example of template for information storage requirements. In particular, this template defines the use case *login*. The other types of requirements such as storage information, actor, or interaction are described by using similar templates.



Formatted Templates

FR-01	Login	
Description	Authentication to allow access to the checkout process	
Actors	Use case actor	
	AC-01. <i>WebUser</i>	
Normal sequence	Step	Action
	1	The system asks for the <i>userID</i> and <i>password</i> and the option to <i>remember both userID and password</i>
	2	The user puts the <i>userID</i> and the <i>password</i>
	3	The <i>userID</i> and the <i>password</i> are <i>checked</i>
	4	The <i>userID</i> and the <i>password</i> is stored if the field <i>remember</i> is true
	5	Access to checkout is allowed
Exceptions	Step	Action
	4	The user is not registered, so the user executes FR-02
	4	The <i>userID</i> or the <i>password</i> are not valid, continue with step 1

Figure 2.9 Example of requirements specification in NDT

Main Requirements Specification Limitations: The specification of requirements in NDT is mainly focused on the use of textual templates. These templates are very suitable to describe requirements such as data requirements or requirements related to user types. However, in the development of complex Web applications there may be a lot of different possibilities of navigation. To specify all of them by using just textual

2. State of the Art

templates is not always easy. In the same way, the use of textual templates (instead of other diagrammatic tools such as the UIDs of OOHDM or the activity diagrams of UWE) makes it difficult to obtain from the requirements specification a complete vision (a “big picture”) of the Web application navigational aspect. Finally, in complex Web applications, the amount of textual templates that must be defined may result very difficult to manage and maintain, and as the own authors of NDT states [Escalona et al. 2006], templates are not easy to complete as they require intensive interviews.

Tool Support for RE activities: NDT is supported by the NDT tool [Escalona et al. 2003]. This tool provides fully support to the whole development process of NDT: it provides support for the specification of requirements, the analysis of them and the generation of HTML-based prototypes.

Guidelines for Obtaining Conceptual Models: NDT proposes a strategy to systematically derive both partial conceptual models (an UML class diagram) and partial navigational models from requirements specifications. This strategy is applied in the analysis phase. The way in which these conceptual models are derived from requirements is detailed in [Escalona 2004]. The derivation algorithm is implemented by the NDT tool, which allows us to automatically apply it. The main drawback of this approach is that the obtained conceptual models (and specially the navigational one) are defined with the main purpose of analyzing and validating requirements. In this context, only basic conceptual primitives are systematically derived.

To overcome this drawback, the authors of NDT together with the authors of UWE have developed an UML profile for Web requirements (WebRE). Thus, in recent works related to NDT [Koch *et al.* 2006] [Segura *et al.* 2007] we can see how it fits WebRE. In this context, the model-to-model transformations presented in these works¹⁶ for obtaining conceptual models from requirements specifications can be applied to NDT-based requirements specifications. The conceptual model that is obtained however is the one proposed by UWE. The main drawback of this approach is that mode-to-model transformations are still not supported by a tool.

2.2.10 Proposals of new Development Processes for Web Applications

The approaches presented above are mainly focused on providing mechanisms such as models or tools that support the development of Web applications. Other approaches, however, have focused their efforts on providing new development processes for Web applications, indicating which activities must be performed but prescribing the use of existing techniques for supporting these activities. From this type of approaches it is worth mentioning the following ones:

¹⁶ Which have been already introduced in the analysis of UWE

2. State of the Art

OO/Pattern Approach: This approach to hypermedia collection design was presented by Thomson, Greer and Cooke in 1998 [Thomson et al. 1998]. It proposes to use both, an OO-design and the application of patterns for the navigational and presentational design. The use of patterns has well-known advantages, such as that the process is well defined, documentation can be reused and maintenance is easy.

This approach is divided into six phases: use case design, conceptual design, collaboration design, dictionary definition, navigational design and implementation. Requirements are handled in the first phase where the use of use case diagrams is prescribed to specify the application requirements.

HFPM: Hypermedia Flexible Process Modelling (HFPM) was developed by Luis Olsina in 1998 [Olsina 1998]. This approach proposes thirteen phases for handling aspects such as requirements modelling, project planning, conceptual and navigational modelling, abstract interface modelling, validation, product quality assurance, documentation, etc.

As far as the specification of Web application requirements, it is proposed to be performed by means of:

1. A problem description. HFPM indicates that natural language can be used in this description. This description constitutes a textual requirements specification.
2. A use cases model in order to refine the textual requirements describe above.
3. A glossary model. It is used to extract essential problem domain keywords. HFPM proposes to describe these keywords in a classified list written in natural language.
4. A user interface model created by using sketches or prototypes. This model is used in the presentation of drafts to the customer.

RNA: The Relationship Navigation Analysis (RNA) approach was developed by Joonhee Yoo and Michael Bieber in 1998 [Yoo & Bieber 2000]. This approach is based on the analysis of relationships among the different elements of interest in the Web application domain. This analysis of relationship can help developers to correctly design the navigational structure (nodes connected through links) of Web applications.

The development process of this approach is divided into five phases:

1. *Stakeholder analysis:* The purpose of this phase is to identify the application's audience (stakeholders). Furthermore, analysts also identify and understand the tasks that each type of user might want to perform within the system.
2. *Element analysis:* In this phase, analysts list all the potential elements of interest in the application. RNA classifies these elements into two types

2. State of the Art

depending on the system under analysis. For a new system, these elements may include all subsystems within the application; all processes that users conduct within the application; all internal processes within the application; and all operations that could be invoked. For existing systems (when a legacy system for the World Wide Web is being reengineered), these elements may include all types of items displayed in any on-line display (information screens, forms, documents, and any other type of display), as well as the screens, forms and documents themselves.

3. *Relationship and meta-knowledge analysis*: In this phase, analysts identify relationships for each element of interest in order to obtain a schema of the Web application. RNA proposes a set of predefined types of relationships that need to be considered in this analysis.
4. *Navigation analysis*: In this phase, the navigational aspects of a Web application are defined from the relationships identified between the elements of interest.
5. *Relationship and meta-knowledge implementation analysis*: In this phase, analysts take decisions about how the information identified in the previous phases is implemented in a specific technology

Design-Driven Requirements Elicitation approach: This approach is part of the Web application design-driven process introduced by David Lowe and John Eklund in 1999 [Lowe & Eklund 2002]. They propose a generic process of Web development that is both iterative and utilizes design prototypes to assist the client in exploring possible solutions and hence formulating requirements.

This process presents three phases (evaluation, specification and building) that are iteratively performed in two different cycles: exploration and build. In the first cycle, developers repeatedly build prototypes and explore them with the client, obtaining feedback and distilling from this information on the client needs in order to progressively build a specification. Once a sufficient understanding of the client requirements has been obtained, a build contract can be negotiated and then the build cycle commenced. The build cycle is also iterative, and the client understanding and the specification is likely to continue to change. The development in this case is aimed at actually building the system.

It is worth to remark that this approach has been defined from an exhaustive analysis of “best practices” in the development of Web commercial applications.

GX WebEngineering Method: This method was developed by Jurriaan Souer, Inge Van De Weerd, Johan Versendaal and Sjaak Brinkkemper in 2006 [Souer *et al.* 2006]. The GX WebEngineering Method proposes a development process defined by method engineering. This means that phases of existent development methods are selected in order to define a new one that provides support to the Web Application development. In particular, the selected methods are: an old version of GX which is

2. State of the Art

property of a company in the Netherlands¹⁷, UWE [Koch 2000] and the Unified Process [Jacobson et al. 1999].

GX is divided in six project phases: Acquisition, Orientation, Definition, Design, Realization and Implementation. For every phase three routes exist: a standard, complex and migration route. Depending on the route different activities are proposed for each phase. The use of each route depends on the complexity of the Web applications. In every route, requirements are described by starting with a goal-oriented description. From this description, each route proposes to continue with different types of techniques such as use cases, description of domain terms, description of user types, etc.

2.2.11 Other Studied Approaches

In order to perform the detailed analysis presented in this chapter, we have studied other approaches that have not been included because they do not consider RE activities in their development process. These approaches are mainly focused on providing conceptual models for Web applications. From these approaches it is worth mentioning the following ones:

HDM: The Hypermedia Design Method (HDM) was developed by Garzotto, Paolini and Schwabe in 1993 [Garzotto et al. 1993]. It is one of the first methods that was developed to define the structure and interaction in hypermedia applications. HDM is based on the Entity-Relationship model [Chen 1976], but extend this model by introducing new primitives to capture what HDM calls the navigational semantics. These primitives are entities (an extended version), components, perspectives, units and links. Currently, HDM is little used because it is based on the structural paradigm and nowadays the object oriented paradigm has gathered strength. However, approaches such as OOHDM, RMM or W2000 were inspired by the ideas of HDM.

RMM: The Relationship Management Metodology (RMM) was developed by Tomas Izsakowitz, Arnold Kamis y Marios Kounfaris in 1995 [Isakowitz et al. 1995]. It is based on both the E-R model and HDM. Taking these models as sources RMM proposes another model (Relationship Management Data Model, RMDM) in which domain objects, relationships between these objects and navigational mechanisms (such as links, grouping (menus), indexes or guided tours) are described. This approach proposes a development process divided into seven phases: entity-relationship design, slice design, navigational design, user interface design, protocol conversion design, run-time behaviour, and construction and testing. RMM is supported by the Relationship Management CASE Tool (RMCASE) [Diaz et al. 1995]. However, this approach is currently little used because it is based on the E-R paradigm.

¹⁷ <http://www.gx.nl/>

2. State of the Art

MAC-WEB: The MacWeb Hypermedia Development Environment was developed by Nanard and Nanard in 1995 [Nanard & Nanard 1995]. Authors emphasize that communication with users is one of the most important aspects of Web applications. Thus, the development of Web applications is mainly focused on the user interface design. This approach considers this design process from two dimensions: methods steps and mental processes. From the first dimension, formal design techniques to produce the design should be prescribed. Five steps are proposed to do this: concepts elicitation, navigational model, abstract interface, implementation model and testing. From the second dimension, how people actually conduct the design process should be observed. To do this, four steps are proposed: generating material, organizing and structuring, reorganizing and updating and evaluating. Finally, it is worth to remark that designers switch from mental process to method steps as there are not predefined transition rules between the activities or steps.

EORM: The Enhanced Object-Relationship Model (EORM) was developed by Danny Lange in 1996 [Lange 1996]. This approach is divided into three main phases: analysis, design and implementation. In the first phase, an object oriented model is defined by following the OMT approach [Rumbaugh et al. 1990]. In the second phase, this model is extended by adding additional semantics to the relationships between objects in order to create navigation links. These links are created from a library of links that contains a set of predefined types of links. Finally, in the third phase, the information captured in models is translated into a specific implementation technology. This approach is supported by the ONTOS Studio tool, which have been developed by the author of EORM.

Araneus Project: The work presented in this project was developed by Giasalvatores Mecca, Paolo Atzeni and Paolo Merialdo in 1999 [Mecca *et al.* 1999]. This work is focused on the management of data bases from a Web environment. This work consists basically in the introduction of hypertext views in the entity-relationship model in order to indicate how data is provided throughout the Web.

WebComposition: This approach was developed by Hans-Werner Gellersen and Martin Gaedke in 1999 [Gellersen & Gaedke 1999]. The main goal of this approach is to facilitate the maintenance of Web applications. To do this, they propose the development of Web applications from components. The WebComposition development process starts by defining components that can represent individual links, anchors or other complete resource, such as a HTML page or a Perl Script generating a HTML page. Next, they are composed to develop the Web application.

WUML: Web Unified Modelling Language aims at the methodological support of Web application development with special focus on ubiquity. It was developed by Kappel, Pröll, Retschitzegger and Schwinger in 2001 [Kappel *et al.* 2001]. Authors consider that Web applications should be designed from the start taking into account not only its hypermedia nature, but also the fact that they must run on a variety of platforms such as mobile phones, PDAs, full-fledged desktop computers, and so on. In this context, they propose a model to describe Web applications in which

2. State of the Art

customization plays a main role. Customization is proposed as a uniform mechanism to provide adaptability of a ubiquitous web application.

2.3 Conclusions

We have presented a brief analysis of the main approaches published in the literature that provide support for the development of Web applications. In this analysis, we have focused on the requirements specification techniques that these approaches propose. Then, we have presented first (from Section 2.1.1 to Section 2.1.9) those approaches that support the development of Web applications and moreover consider the requirements specification in its development process. These approaches have been presented chronologically according to the year in which they were presented. Next, in Section 2.1.10 we introduce, in a schematic way, several proposals of Web development processes. These proposals only focus on providing development processes and do not provide new techniques for supporting the development of Web applications. Finally, Section 2.1.11 introduces a brief overview of approaches that support the development of Web applications but ignore the phase of requirements.

The main conclusions of this analysis are next presented. They are organized in three parts: first, we analyze the results of studying the techniques proposed to specify Web application requirements; second, we summarize the conclusions obtained from analysing the tools that are developed for specifying requirements; third, conclusions related to the guidelines that exist to facilitate the construction of Web application conceptual models are studied.

Requirements specification techniques: We have seen how approaches such as HDM, RMM or ORM belong to a first generation of Web engineering approaches that only provide techniques for the activities of modelling and design of Web applications. Requirements are not considered. Even OOHDM in its first versions proposed a development process that starts in a conceptual modelling phase.

In a second generation of Web engineering approaches, requirements start to be considered. However, they are handled in different ways. Some approaches such as the Conallen's approach, WebML, WSDM or OOH are mainly focus on the definition of conceptual models for Web applications and they prescribe the use of traditional techniques for requirements specification. These techniques are mainly focused on the use of use cases diagrams. In this context, we have seen how several works have been presented to demonstrate that use cases are few appropriate for the specification of navigational requirements.

Other approaches such as OOHDM (in its recent versions), UWE, W2000 or NDT also propose the use of use cases in the requirements specification. However, these approaches propose additional techniques for complement use cases in order to

2. State of the Art

properly support the navigational requirements. OOHDM and UWE provide diagrammatic tools to describe use cases in detail. The main problem of this solution is that they provide a narrow view of navigational requirements since they focus on use cases individually. W2000 provides two types of use case models in order to identify navigational capabilities. However, techniques for describing use cases in detail are not provided. Finally, NDT extends use cases with textual templates that allow us to specify different types of requirements. This solution makes it difficult to visualize the “big picture” of the navigational requirements of a Web application.

SOHDM is the only approach that provides a specific requirements specification technique for Web applications that is not based on use cases. This approach is based on scenarios and it proposes a proprietary notation to describe them. However, this notation is mainly focused on the specification of transactional requirements and we have seen how the specification of navigational requirements may result not always easy.

As far as the proposals of Web application development processes (OO/Pattern approach, HFPM, RNA, Lowe and Eklund’s process and GX WebEngineering Method) all of them consider the specification of requirements. However, either they prescribe the use of traditional techniques for requirements specification such as use cases or do not prescribe the use of any technique.

Tools for specifying Web application requirements: As far as tools that support the requirements specification of Web applications, only UWE and NDT provide them. UWE is supported by the ArgoUWE tool. This tool provides graphical support to create the diagrams proposed by UWE (use case diagrams and activity diagrams) in order to specify Web application requirements. NDT is supported by the NDT tool. This tool provides graphical support in order to create use case diagrams and the formatted textual templates.

Other approaches such as WebML and OOH are also supported by a tool. However, WebRatio and Visual Wade (tools which support WebML and OOH respectively) are mainly focused on providing graphical interfaces for the definition of Web conceptual models (and the generation of code from these models) and provide little support to the specification of requirements.

Guidelines for the creation of conceptual models: As general rule, Web engineering approaches leave the way in which requirements are translated to a conceptual model in hands of analysts’ experience and skills. Guidelines for supporting the creation of Web application conceptual models are only provided by three approaches: OOHDM, UWE and NDT.

OOHDM explains these guidelines in natural language, running the risk of resulting ambiguous. Furthermore, no tools are provided that helps analysts to follow the proposed guidelines. UWE and NDT provide a common Web requirements meta-

2. State of the Art

model and a set of model-to-model transformations that taking as source a description based on this meta-model allows us to systematically obtain conceptual models of Web applications. These model-to-model transformations are defined in the QVT Relations standard. There is no tool however that supports these transformations.

To conclude this section, we can claim that:

- There exist few techniques defined specifically for the specification of Web application requirements. Furthermore, the newly created techniques can be considered still young techniques and a lot of research work can be performed in order to improve them. In particular, we should guide our efforts to improve the specification of navigational requirements.
- There exist few tools that support the requirements specification of Web application requirements. In this context, almost all the models proposed by the different approaches to specify Web application requirements must be manually created.
- Once requirements are specified there is little methodological support that helps analysts in the creation of the conceptual schema that properly satisfy the requirements specification. Logically, few or none tools are provided to support this aspect.

Chapter 3

*“Things should be made as simple as possible,
but not any simpler”*

Albert Einstein.
(German theoretical physicist, 1879-1955)

3 A Web Application Development Process

In this chapter, we introduce a model-driven development process for the Web application development. This process constitutes an extension of the development process of the Web engineering method OOWS [Fons et al 2003]. In this extension, we introduce support for RE activities. This process is presented in Section 3.2.

From among the mechanisms provided by the OOWS method it is worth to remark a powerful strategy of automatic code generation. This strategy allows us to automatically obtain the software artefacts that are equivalent to the different abstractions defined in the OOWS conceptual models. We combine this strategy with a set of model-to-model transformations in order to define a mechanism that allows us to automatically obtain Web application prototypes from requirements specifications. Then, in order to guide analysts in the RE activities we present an iterative RE process for Web applications that supports rapid prototyping from requirements specifications. It is presented in Section 3.3.

3. A Web Application Development Process

In order to better understand the different software processes that are presented in this chapter, a summarized description of the notation that is used is previously presented in Section 3.1

3.1 Describing Software Processes. The SPEM Notation

In order to define software development processes several approaches such as PIE [Cunin et al. 2001], OPEN [OPEN 1997] or SPEM [SPEM 2002] have been proposed. In this thesis, we use the Software Process Engineering Metamodel (SPEM). We use SPEM v1.1 because it provides us with enough abstractions for properly describing the process that we propose. Furthermore, it is the current standard proposed by the OMG [OMG].

SPEM allows us to define a software process by means of a set of *activities* [Jacobson et al. 1995]. Each of these activities is performed by one or more *process roles*. After the performance of each activity, one or more *work products* can be obtained (output work products of the activity). An activity can also require some work products in order to be performed (input work products of the activity). SPEM proposes different types of work products. In this thesis, we use three types of work products: *Documents*, *UML-based models* and *Executables*. The notation proposed by SPEM in order to represent all these software process elements is presented in Figure 3.1A.

Additionally, SPEM proposes the use of UML 2.0 activity diagrams [UML] in order to define sequences of activities as well as their input and output work products. In this case, nodes in activity diagrams represent activities or work products. Arcs in activity diagrams represent: (1) a sequence of activities (depicted by solid arrows) if both the source and the target of the arc are activities or (2) the output or the input work products of an activity (depicted by dotted arrows) if the target or the source of the arc is a work product. Solid arrows are also used to indicate the initial activity within a software process. To do this, these arrows connects the initial activity with the process role that performs it. Figure 3.1B shows some representative examples of these arcs.

SPEM also allows us to associate process elements with *guidance* elements. Activities are associated to guidance elements in order to indicate detailed information about how practitioners may perform the activities. In this thesis, these guidance elements refer to the different chapters/appendices of the thesis that introduce information related to the activity. We also represent these guidance elements in the activity diagram. We connect activities to guidance elements by means of dashed arrows. In Figure 3.1B we can see an example of an activity with guidance.

3. A Web Application Development Process

Finally, it is worth to remark that the notion of *process* can also be represented by means of the SPEM notation (see Figure 3.1A). In this thesis, we use this notion in order to include predefined processes (i.e. predefined sequences of activities with their input and output work products as well as the corresponding process roles and guidance elements) in the definition of more complex processes (we include them as sub-processes). A sub-process can be connected either to activities by means of solid arrow or to work products by means of dotted arrows. In both cases, this connection implies to connect activities and work products of the complex process to the initial or final element of the sub-process, depending on whether the sub-process constitutes the source or the target of the arc, respectively. We graphically illustrate this aspect in Figure 3.1C.

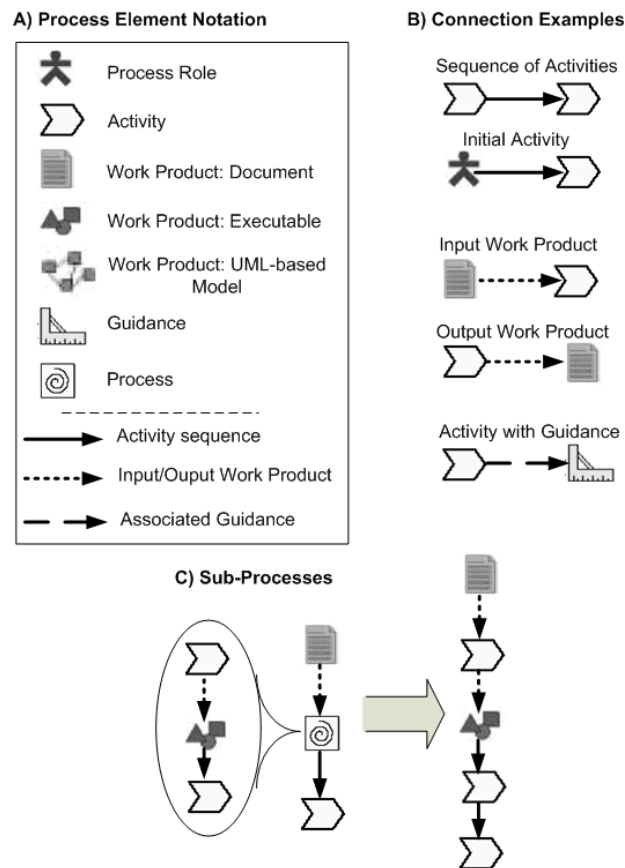


Figure 3.1 SPEM Notation

3. A Web Application Development Process

3.2 The OOWS development Process with RE support

In this section, we present the OOWS development process extended with support for RE activities. The OOWS development process has been already presented in the published literature [Fons *et al.* 2003] [Pastor *et al.* 2005]. In these works however this process has been always presented in an intuitive way by using an informal notation. Thus, in addition of extending the OOWS development process to support RE activities, we provide, as an additional contribution, the description of the OOWS development process with a specific notation for software process definition such as SPEM.

The OOWS development process is based on the Model Driven Development (MDD) paradigm [Mellor *et al.* 2003]. MDD proposes to develop software from models. It proposes to create a model of a system in order to transform it into the equivalent software product. The software development's primary focus and products are models rather than computer programs.

MDD is the next step in the work in which researchers have been working during years: Raising the abstraction level at which software engineers write programs. The first FORTRAN compiler was a major milestone in computer science because, for the first time, it let programmers specify *what* the machine should do rather than *how* it should do it. MDD is a natural continuation of this trend and models constitute a way of specifying what the machine should do at a higher level of abstraction.

Models are used to better manage complex system descriptions by using different models to capture the different aspects of the solution. The OOWS method proposes five models in order to capture the static and dynamic structure of Web applications as well as the presentation and navigational aspects. See Appendix B in order to obtain information about the OOWS conceptual schema.

For the MDD vision to become reality, tools that support model transformations must be provided. These model transformations imply transformation between models (model-to-model transformations) and derivation of code from models (model-to-code transformations). The OOWS method proposes an automatic code generation strategy supported by the OOWS CASE tool in order to generate code from the models of its conceptual schema. Additional information about this strategy can be also found in Appendix B.

Thus, the development process of the OOWS method is divided in two major steps (see lower side in Figure 3.2): (1) *System Specification* in which the OOWS conceptual schema is created and (2) *Solution Development* in which a strategy oriented towards generating the software components that constitute the solution (the final software product) is defined.

3. A Web Application Development Process

The work presented in this thesis introduces a RE approach for supporting the development of Web applications. RE activities are generally related to the earliest stages in the software development life cycle motivated by the evidence that requirements errors, such as misunderstood or omitted requirements, are more expensive to fix later in project lifecycles [Boehm 1981] [Nakajo & Kume 1991]. Thus, we introduce the step *Requirements Analysis* as initial step of the OOWS development process (see upper side in Figure 3.2). In this step, we propose to create the requirements model presented in this thesis (further explained in Chapter 4).

Furthermore, a strategy based on model-to-model transformations is also introduced in order to derive OOWS conceptual models from the requirements model. This strategy is defined from the traceability rules presented in Chapter 5. The strategy to automatically apply transformations is explained in Chapter 6. The result obtained after the application of the model-to-model transformations is a skeleton of the OOWS conceptual schema that can be used as base to perform the System Specification step.

Figure 3.2 shows the OOWS development process extended with support for RE activities. Extensions are highlighted in grey. They constitute the main contributions of this thesis. In this process, three process roles are defined: Model Transformation Tools, Analysts and the OOWS CASE tool. We include tools as process roles because they are able to perform their associated activities in an automatic way.

The process starts with the Requirements Analysis activity where analysts create a requirements specification from the customer information. This requirements specification constitutes the output work product of the Requirements Analysis activity. The guidance element that supports analysts in the performance of this activity is the iterative and prototyping RE process presented in Section 3.3.1.

Once the requirements specification is obtained it constitutes the input work product of the activity Model-to-Model Transformation. This activity is automatically performed by a set of model transformation tools. The guidance elements that support this activity are the traceability rules presented in Chapter 5 as well as the tool-supported strategy for automatically applying model-to-model transformations presented in Chapter 6. The output work product of the Model-to-Model Transformation activity is a skeleton of the OOWS conceptual schema.

The next activity to be performed in the OOWS development process is the System Specification. In this activity, analysts take as base the skeleton of the OOWS conceptual schema and complete it according to the requirements specification. In this context, this activity has two input work products: the skeleton of the OOWS conceptual schema and the requirements specification (obtained from the Requirements Analysis activity). The guidance element of this activity is the Appendix B that explains how the OOWS conceptual schema is constructed. The output work product of this activity is a fully defined OOWS conceptual schema.

3. A Web Application Development Process

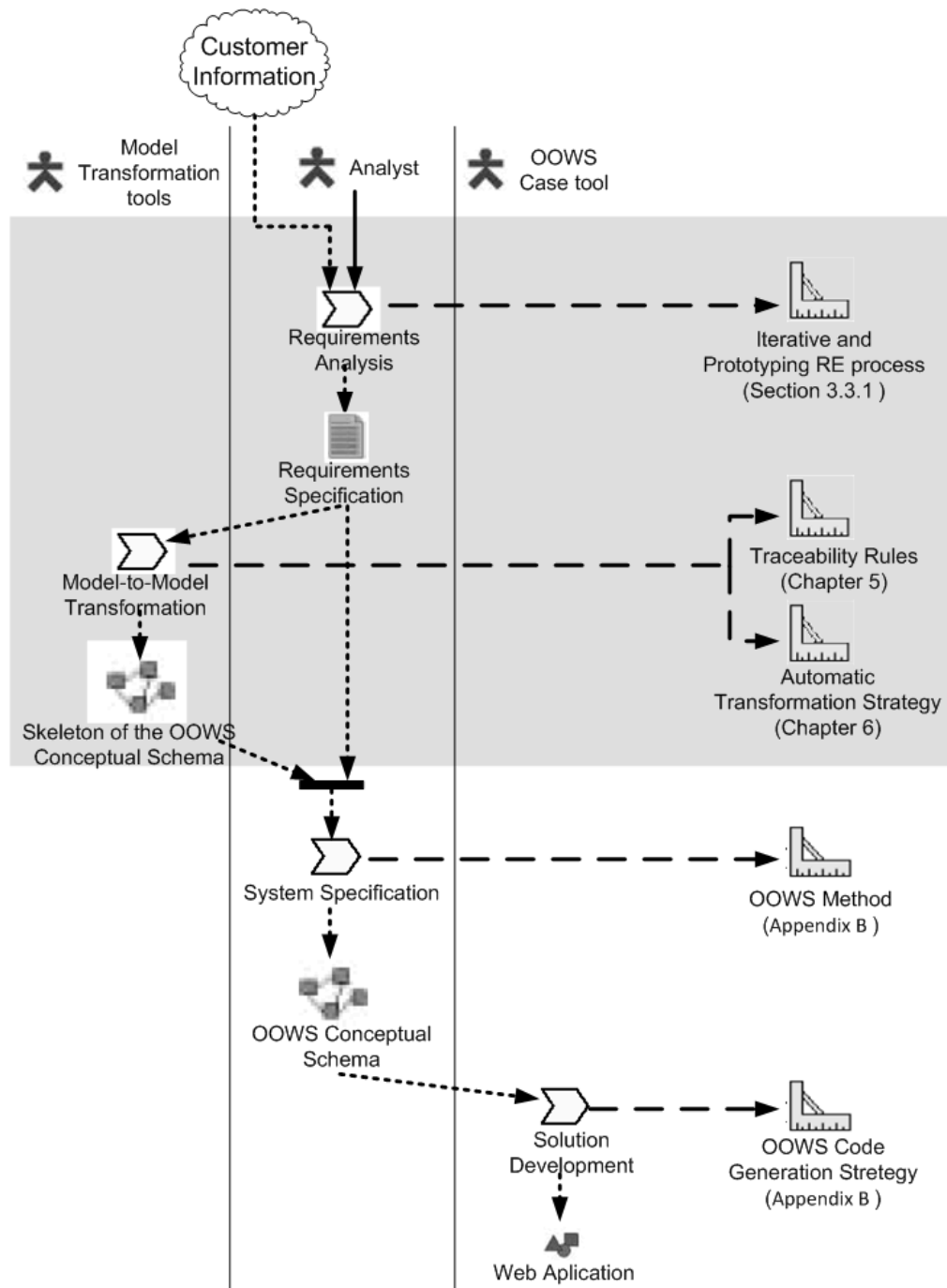


Figure 3.2 The Extended OOWS development process

3. A Web Application Development Process

Finally, the fully defined OOWS conceptual schema constitutes the input work product of the next and last activity in the OOWS development process: Solution Development. The process role in this activity is the OOWS CASE tool. This tool takes the OOWS conceptual schema as source and automatically transforms it into the equivalent software product that, in this case, is a Web application. This Web application constitutes the output work product of the activity Solution Development. The guidance element that supports this activity is the Appendix B, where the OOWS code generation strategy is described. The OOWS CASE tool is also presented in this appendix.

3.3 Improving the Requirements Analysis Activity throughout Prototyping

According to the OOWS development process, Web applications are automatically generated from the OOWS conceptual schema in the Solution Development activity. These Web applications are implemented by following a three tier architecture that is made up of (see Appendix B for more detailed information): (1) a Presentation tier that implements the graphical user interface, (2) an Application tier that implements the structure and the functionality of the classes in the conceptual schema and (3) a Persistence tier that implements the persistence and the access to data.

In order to automatically generate these three-tier based Web applications, the OOWS conceptual schema must be properly built to describe the different aspects of Web applications. This means that every conceptual model included in the OOWS conceptual schema must be fully defined. These models are five (as we explain in Appendix B): the structural model that captures the static structure of the system, the dynamic and functional models that capture the dynamic structure (the system behaviour), the navigational model that defines the navigational structure of the Web application, and finally, the presentation model that indicates how the information is shown.

To fully define these models, the System Specification activity is performed by Analysts (see Figure 3.2). However, analysts do not create these models from the scratch but they take as base a skeleton of the OOWS conceptual schema that is obtained from the requirements specification (by means of the model-to-model transformation activity, see Figure 3.2). This skeleton of the OOWS conceptual schema is made up of a partial set of conceptual primitives that can be systematically derived from requirements. These conceptual primitives are, as we further explain in Chapter 5, those that allow us to partially define both the structural model and the navigational model.

Although the structural model and the navigational model are partially derived, they contain enough information to implement Web application prototypes from them.

3. A Web Application Development Process

These Web application prototypes partially implement the presentation tier (by means of a set of interconnected Web pages implemented from the navigational model) and the persistent tier (by means of a data base schema implemented from the structural model). Furthermore, an intermediate code that connects Web pages to the data base can be also implemented. More detailed information about these prototypes can be found in Appendix B.

In this context, it is possible to apply the OOWS code generation strategy directly to the skeleton of the OOWS conceptual schema. In this case, we do not obtain full Web application (as happens in the activity Solution Development) but we obtain operative Web application prototypes. Thus, if we consider that:

- (1) Skeletons of the OOWS conceptual schema are automatically obtained from requirements specifications throughout model-to-model transformations (see Chapter 6)

and

- (2) The OOWS code generation strategy allows us to automatically obtain Web application prototypes from skeletons of the OOWS conceptual schema (see Appendix B)

Then:

We can automatically obtain Web application prototypes directly from requirements specifications.

The process for constructing Web application prototypes from requirements specifications is shown in Figure 3.3. In this process, two process roles are defined: Model Transformation Tools and OOWS CASE tool.

The process starts with the activity Model-to-Model transformation that is performed by the Model Transformation Tools. These tools take a requirements specification (that is defined as input work product of the activity) and derive a skeleton of the OOWS conceptual model from it. The skeleton constitutes the output work product of this activity. The guidance elements that support this activity are the traceability rules presented in Chapter 5 as well as the tool-supported strategy for automatically applying model transformations presented in Chapter 6.

The skeleton of the OOWS conceptual model is the input work product of the next activity: Prototype Generation. This activity is performed by the OOWS CASE tool. This tool automatically generates a Web application prototype from the skeleton of the OOWS conceptual model. The obtained prototype constitutes the output work product of this activity. The guidance element that supports this activity is the OOWS

3. A Web Application Development Process

code generation strategy that is described in Appendix B. The OOWS CASE tool is also presented in this appendix.

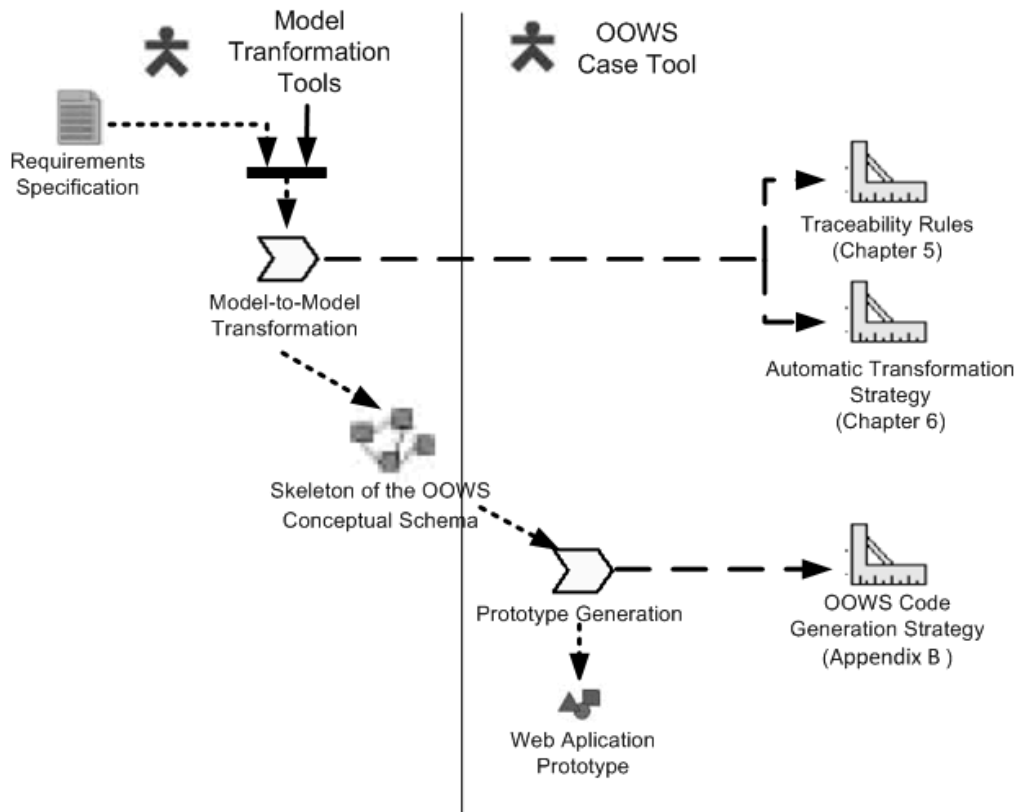


Figure 3.3 The prototype construction process

This prototype construction process allows us to obtain Web application prototypes directly from requirements specification. We introduce next how this process is used to define an iterative and prototyping RE process for Web applications.

3.3.1 An Iterative and Prototyping RE Process for Web applications

A RE process is a structured set of activities which are followed to derive, validate and manage the requirements of a system. There is no a universally accepted RE process. The definition of a good RE process depends on the organization where the system must be introduced. There are many ways to organise RE processes and they do not transfer well from one organization to another. However, we would normally expect a good RE process to include the following activities [Sommerville & Sawyer 1997] [Sawyer & Kotonya 2001]:

3. A Web Application Development Process

1. *Requirements Elicitation*: The system requirements are discovered through consultation with stakeholders, from system documents, domain knowledge and market studies. This activity is also known as requirements capture, requirements discovery or requirements acquisition. Some techniques for performing this activity are: interview [Durán *et al.* 1999] [Pan *et al.* 2001], JADs [Livesey & Guinane 1997], Brainstorming [Raghavan *et al.* 1994], Questionnaire and Checklists, Concept Mapping [Pan *et al.* 2001], etc.
2. *Requirements Analysis and Negotiation*: The requirements are analysed in detail, and there should be some formal negotiation process involving different stakeholders to decide on which requirements are to be accepted. The main objective of requirements analysis and negotiation is to establish an agreed set of requirements which are complete and consistent. These requirements should be unambiguous so that they can be used as a basis for further system development. During this process, analysts normally discover missing requirements, requirements conflicts, ambiguous requirements, overlapping requirements and unrealistic requirements.

Requirements analysis and negotiation are concerned with the high-level statement of requirements elicited from stakeholders. Requirements engineering and stakeholders negotiate to agree on a definition of the system requirements. In this context, the construction of abstract descriptions that are amenable to interpretation is a fundamental step in requirements analysis activity. Developing these abstract descriptions usually reveals further contradictions and incompleteness in the requirements. In this way, this activity is many times considered as a part of the requirements specification activity since these abstract descriptions become usually the final requirements catalogue.

3. *Requirements Specification*: It is the activity by means of which the information obtained in the previous activity is defined in a requirements catalogue (also known many times as requirements specification or requirements model). This activity is also known as requirements definition or requirements documentation. For the requirements specification activity, many techniques have been proposed: Natural Language, Glossary and Ontology, Templates, Scenarios [Liu & Yu 2001], Use CASE Modelling, Formal description, Task Analysis, etc.
4. *Requirements Validation*: Through requirements validation the requirements specification is checked to correspond to the user's needs and the customer's requirements. The main objectives of requirements validation is discover possible problems with the requirements. The process should involve system stakeholders, requirements engineers and system designers. Some of the most used techniques in the validation of requirements are: Review or Walk-through, Audit, Traceability Matrix, Prototyping, etc.

3. A Web Application Development Process

The RE Process Proposed in this Thesis

In this section, we introduce the RE process that is proposed to be used in the context of this thesis. It is shown in Figure 3.4. This process is based on the one presented in [Lowe & Hall 1999] which is divided in three main activities: Requirements Elicitation, Requirements Specification (the requirements analysis and negotiation activity is considered to be part of the specification activity) and Requirements Validation. We extend this RE process by introducing the prototype construction process as part of it. This aspect allows us to support rapid prototyping in RE activities.

The RE process includes two process roles: Clients and Analysts. The process starts with the activity Requirements Elicitation in which analysts identify the users' needs in order to define a draft statement of the Web application requirements. This draft statement constitutes the output work product of the activity. The main contributions of this thesis are not focused on improving the elicitation of Web application requirements. Then, we prescribe the use of traditional techniques to support this activity.

The draft statement of requirements is the input work product of the next activity: Requirements Specification. In this activity, analysts analyse the draft statement of requirements in order to create a requirements catalogue which constitutes the output work product of the activity. To do this, the task-based requirements model for Web applications that is presented in Chapter 4 is used. This requirements model constitutes the guidance element associated to the activity Requirements Specification.

The requirements catalogue obtained in the Requirements Specification activity is the input work product of the prototype construction process. As we have seen above, this process automatically generates a Web application prototype from the requirements catalogue.

Both the Web application prototype and the requirements catalogue constitute the input work products of the last activity in the RE process: Validate Requirements. In this activity, clients validate the requirements catalogue defined by analysts. To do this, we propose clients to interact with the Web application prototype that is obtained by following the prototype construction process.

3. A Web Application Development Process

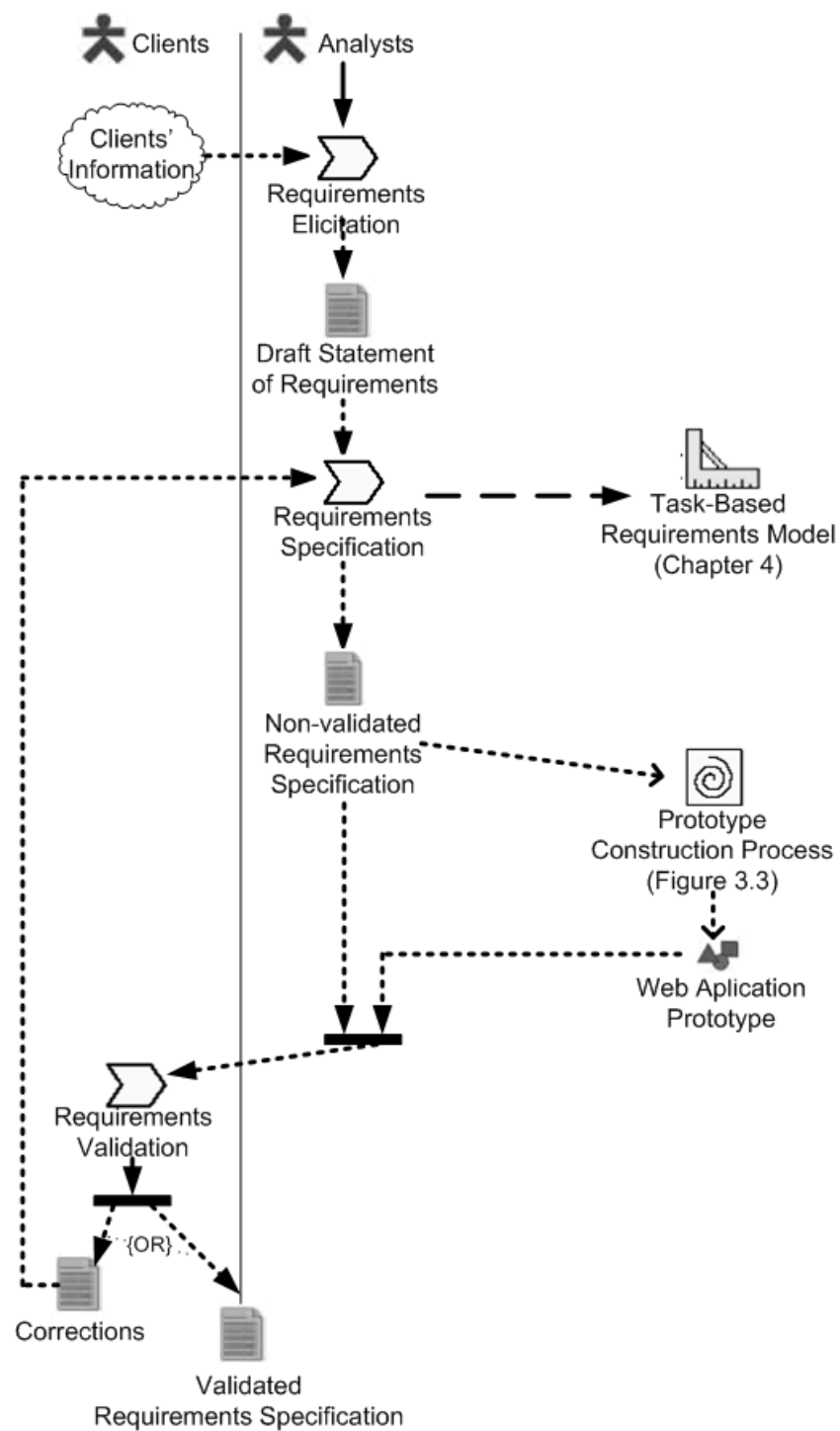


Figure 3.4 A RE process for Web applications

3. A Web Application Development Process

The Validation activity has two different output work products:

- (1) A set of corrections, if inconsistent requirements or mistakes are detected; in this case, analysts must modify the requirements catalogue performing the RE process again from the activity Requirements Specification.
- (2) A validated requirements catalogue, if all requirements are correct; in this case, the RE process finishes. This requirements catalogue constitutes the input work product of next activities in the whole development process (Model-to-Model Transformation and System Specification, see Figure 3.2)

This RE process supports analysts in the performance of the activity of Requirements Analysis (see Figure 3.2). Furthermore, this process allows analysts to actively involve clients in the development process by allowing clients to validate requirements throughout the interaction with the Web application prototype. This aspect also facilitates the creation of a more complete and precise requirements specification since requirements often emerge only after users have had an opportunity to view and provide feedback on prototypes [Jeenicke et al. 2003] [Lowe 2003].

Furthermore, the use of the prototype construction process in RE activities provides us with the following benefits:

- A software product is already obtained in the RE stage without programming effort. Since it is automatically implemented by tools, analysts do not need to interact with programmers in order to implement the Web application prototype.
- Analysts can obtain as many Web application prototypes as they need. When requirements change Web application prototypes can be automatically re-implemented.
- The prototype can be obtained as soon as the requirements model is constructed. In this context, clients got excited when they interact with a software product that partially supports their needs soon afterwards they have been interviewed by analysts.
- Traceability aspects. According to the prototype construction process, Web application prototypes are implemented directly from the requirements catalogue by following a set of precise and clear steps. In this context, when clients detect some mistakes in the Web application prototype we can apply these steps in the inverse way in order to detect the requirements that are wrong. By analysing the transformation patterns applied by the OOWS code generation strategy we can identify the OWS conceptual primitive from which the wrong software artefact has been generated. Next, by analyzing the

3. A Web Application Development Process

correspondences between models that define the model-to-model transformations we can identify from which requirement the OOWS conceptual primitive has been derived.

3.4 Conclusions

In this chapter, we have extended the development process of the Web engineering method OOWS by introducing a new stage for handling requirements. In order to properly perform the activities associated to this stage we have moreover defined an iterative and prototyping RE process for Web applications.

This RE process proposes the use of the task-based requirements model presented in the next chapter in order to specify Web application requirements. Furthermore, in order to automatically obtain Web application prototypes from requirements specification it proposes the use of the model-to-model transformations presented in Chapter 6 together with the OOWS code generation strategy presented in Appendix B.

Chapter 4

“Metaphors are a way to help our minds process the unprocessable. The problems arise when we begin to believe literally in our own metaphors”

Dan Brown.
(American writer, 1964-)

4. A Task-Based Requirements Model for Specifying Web Applications

In this chapter, we present a requirements model to capture Web application requirements. This model is based on the *task* metaphor, which is widely accepted for the capture of functional requirements [Lauesen 2003]. It has been extended to capture the navigational aspect of Web applications. We extend traditional task descriptions by introducing information about the interaction between the user and the system. Furthermore, task descriptions are combined with a technique based on templates in order to capture the data that the system must store.

According to the requirements classification presented in Chapter 2, the requirements model that is introduced in this chapter allows capturing three types of Web application requirements: data, transactional and navigational requirements.

Web application requirements are specified by identifying first the tasks that users must perform with the Web application. Next, we describe the interaction that users require from the system in order to perform each task. This interaction is

4. A Task-based Requirements Model for Specifying Web Applications

complemented with a set of information templates that describe the information that the system must store to support the performance of each task. In this context, transactional requirements are specified by means of the task descriptions; information templates allow us to capture data requirements; finally, navigational requirements are captured throughout both task descriptions and information templates.

Section 4.1 introduces some background about the task concept and the motivation of using it for the specification of Web application requirements. Section 4.2 introduces the architecture of the task-based requirements model. The meta-model of this model can be found in Appendix A. A tool that supports analysts in the definition of task-based requirements models is presented in Section 4.3. Finally, we conclude this chapter in Section 4.4.

4.1 Tasks. Background and Motivation

4.1.1 The Concept of Task

The concept of task has been used from several areas and then, several definitions can be found throughout the literature. Some examples of these definitions are the following:

- In common language, a task is a piece of work to be done, especially one done regularly, unwillingly or with difficulty [Cambridge 2007].
- From a computer perspective, a task is an operating system concept which refers to "an execution path through the address space". In other words, a task is a set of program instructions that is loaded in memory. In this field, task is also known as process or job [Silberschatz et al. 2004].
- From a project management perspective, a task is specific work item to be undertaken which usually results in partial completion of a project deliverable [Westland 2003].
- From a task analysis perspective, a task is a group of discriminations, decisions and "effector" activities related to each other by temporal proximity, immediate purpose and a common man-machine output [Miller 1956].

By basing on definitions such as the last one, and concretely on the relation between the man and the machine that it associates to a task, the concept of task has been used in different fields of the development of software. The HCI community, for instance,

4. A Task-based Requirements Model for Specifying Web Applications

has used the concept of task to create task models that helps developers to design user interfaces [Diaper 1990]. The Software Engineering community has also adopted a concept of task based on the last definition in order to represent the user needs that a software product should satisfy [Lauesen 2003] [Zhu et al. 2002].

In this thesis, we also use a concept of task that is based on the last definition. Furthermore, as the Software Engineering community has done, we also consider a task to be a representation of specific user needs. In particular, considering that this thesis is focused on the specification of Web application requirements, we consider a task to be:

An activity that users need to perform by interacting with a Web application in order to achieve a specific goal.

Motivation

In general terms, the purpose of a requirements model is to accurately and precisely describe the software system that must support specific users' needs. The software system must be described in such a way that people without high-level training in the notation (stakeholders) can understand and review. The notation should nevertheless be precise enough so that it can be the basis for the conceptual modelling phase. Taking these premises into account, some methods for specifying requirements have chosen a task-based requirements model because:

- Tasks center the requirements modelling process around the user's own experiences and goals instead of other requirements specifications that describe the system's behaviour but ignore the system's context [Lauesen 2003].
- Task-based models are well-understood and can be very expressive [Johnson 1999].
- Tasks descriptions can be used to assess the complexity of the activities that the user is expected to perform in order to anticipate the amount of time required to develop the software [Card *et al.* 1983].

In the particular case of this thesis, all these reasons are also valid to choose a task-based requirements model for capturing Web application requirements. However, our main goal is not only to provide mechanisms for specifying requirements but also to provide mechanisms that allow us to automatically generate Web application prototypes from the requirements model. Then, we need to define a requirements model that accurately describes not only the structural and behavioural aspects of a Web application but also the navigational aspects.

4. A Task-based Requirements Model for Specifying Web Applications

Tasks allow us to define temporal relationships among them. This is another reason for choosing the concept of task in order to capture Web application requirements. This information allows us to systematically extract from the requirements model valuable information that can be used to derive the navigational structure of Web applications. For instance, we can indicate that a task T2 must be started after a task T1 finishes. In this context, we can derive that the navigational structure that allows us to perform T1 must be connected to the navigational structure that allows us to perform T2 by means of a link. We explain this aspect in detail in Chapter 5.

4.1.2 Task Analysis in Software Engineering

Task analysis can be defined as the study of what a user is required to do, in terms of actions and/or cognitive processes, to achieve a system goal [Kirwan & Ainsworth 1992]. Task analysis is a term that covers several techniques. These techniques depend on the purpose for which task analysis is used. Details for 40 tasks analysis techniques are presented in [Kirwan & Ainsworth 1992]. Three of these techniques have a particular relevance to software engineering:

- *Hierarchical Task Analysis (HTA)* is the most commonly used task analysis technique to represent the relationships between tasks and subtasks. HTA breaks down tasks hierarchically from a main goal to sub-tasks, then each sub-task into more and more detail. This type of analysis can be recorded in a tabular form and/or graphically. If recorded graphically it resembles a tree with branches and sub branches as required.
- *Time Analysis (TA)* is used to document the temporal aspects of tasks. It is well suited to represent task dependencies and possible resource problems in performing tasks. It is often difficult to collect the information needed to produce a TA. In absence of observational data estimates or guesses of temporal relations are used.
- *LINK analysis* is one of the simplest techniques. It simply demonstrates the frequency of linkage between tasks. This technique can easily be performed. It is most often based on observational data.

Common elements of these techniques are that all of them describe interactions between humans and their environments in a systematic way. Task analysis therefore is a methodology that is supported by several techniques for helping analysts to collect information, organise it, and then use it to make various judgements or design decisions.

In this thesis, we use a hierarchical task analysis in order to identify the tasks that describe the user needs that a Web application must support.

4. A Task-based Requirements Model for Specifying Web Applications

Motivation

The reasons of using a Hierarchical Task Analysis in the activity of requirements engineering are the following:

- HTA views tasks in a more abstract sense, as a set of interlinked goals, resources and constraints [Shepherd 2001]. It allows us to represent tasks starting from more general tasks and ending with more specific ones. This aspect allows us to attenuate a typical problem in requirements engineering which is traditionally described by the sentence “Clients never know what they want”. With HTA we can start by representing more general tasks that describe what users initially want in a general way, and then extract from this representation more specific information.
- HTA allows us to split the whole set of requirements (the most general task) into smaller groups of interrelated requirements (more specific tasks). This aspect allows us to face the requirements engineering activity by parts. We can start studying the requirements associated to a specific task and once these requirements are completely clear start with the requirements of other task. This aspect also facilitates the management of requirements.
- HTA facilitates the representation of temporal relationships between tasks and, as we have explained above, this aspect helps us to properly capture navigation at the requirements level. For instance, since tasks are hierarchically represented, a relationship can be associated to a parent task instead of being associated to each child task.

4.2 Requirements Model Architecture

The task-based requirements model proposed in this thesis is made up of three main elements: (see Figure 4.1):

1. *Statement of Purpose*. The main purpose of the system is defined by means of a description in natural language.
2. *User Task Description*. First, the different tasks that users need to perform by interacting with the Web application are identified from the statement of purpose. To do this, we propose the definition of a task taxonomy. Next, each leave task of the taxonomy is described from the interaction between the user and the system that is required to perform each task. To do these descriptions, we propose a technique based on UML activity diagrams [UML].

4. A Task-based Requirements Model for Specifying Web Applications

3. *System Data Description*. In this part, we define the data that the system must store taking into account the task descriptions defined above. We also describe it in detail the data that the system and the user exchange in each interaction.

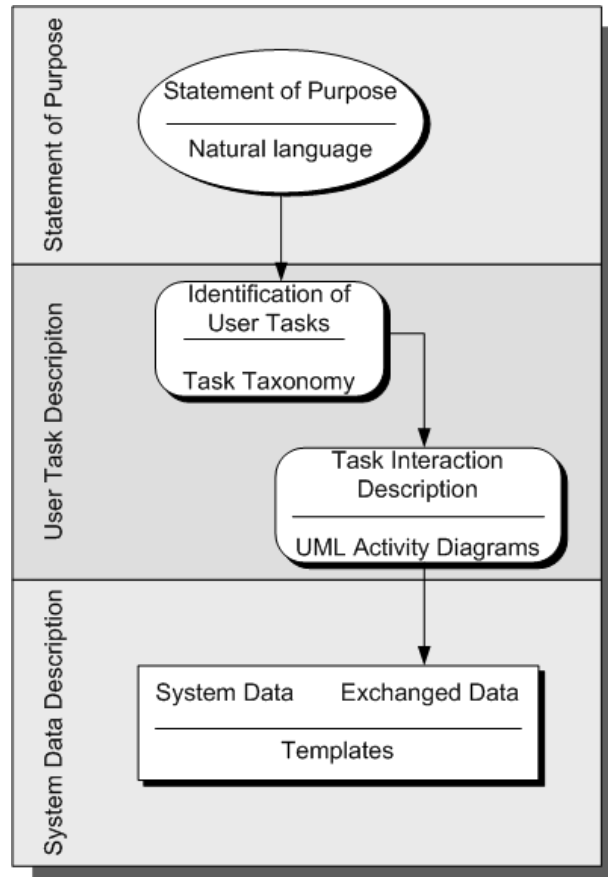


Figure 4.1 Requirements Model Architecture

The fact of capturing Web application requirements in these three different elements allows us to provide a high level of independency among different kinds of requirements. Next, we introduce each element in detail. In order to clearly map the concepts and abstraction mechanisms introduced in the task-based requirements model, the Amazon example (which has been presented in Chapter 2) has been taken as a case study.

4. A Task-based Requirements Model for Specifying Web Applications

4.2.1 Statement of Purpose

An extremely important activity is to define the *Statement of Purpose* (SP) when a system has to be developed. The Statement of Purpose describes the most general service (the main goal) that the system under development provides to its environment. Any external property specification should contain a specification of its main goal and relate this to the objectives of the business.

The SP is a high-level description of the nature and purpose of the Web application. It should be brief and easy to read. The reason for writing it is that customers, users, designers, programmers, testers, and all other stakeholders should keep it in mind during the entire software development process.

According to works such as [Yourdon 1993] or [Insfrán 2003] the SP should basically describe three aspects:

- The type of system that must be developed. In the case of this thesis, the type of Web application that is required by users.
- The purpose for which the Web application is developed. This is a brief description of the main goal that the system must achieve. This description should preferably contain only one sentence.
- A description of the general functionality that is required to achieve the system purpose. It is described by at most two or three sentences.

For instance, the SP of the Amazon example could be as follows:

“The Web application under development is an E-commerce Application. The main goal of the system is to provide support for the on-line purchase of products. To achieve this, the user must be able to consult information about the products, add them to a shopping cart and send purchase orders. Furthermore, tools for the information management must be provided.”

We can see in the example presented above how the first sentence describes the Web application type; the second one indicates the Web application purpose; and the rest of sentences provide a general overview of the Web application functionality.

As explained above, this technique has been already used in works such as [Yourdon 1993] or [Insfrán 2003]. These works use the SP as starting point for detecting the use cases [UML] that describe the requirements of a system. We are inspired by these works and we also use the SP as the initial step in the construction of our requirements model. However, we do not use it to detect use cases. We use the SP as the starting point in the process of identifying the user’s tasks that describe the

4. A Task-based Requirements Model for Specifying Web Applications

requirements of a Web application. We explain this aspect in detail in the next section.

4.2.2 User Task Description

In order to describe user tasks we propose two activities:

Activity 1: Identification of user tasks

Activity 2: Description of tasks

Activity 1: Identification of User Tasks

Next, we explain how to identify and to represent the tasks that users should perform by interacting with the Web application. To do this, we explain how to:

1. Identify the most general task that users can perform
2. Refine the most general task in more specific ones
3. Decide when stopping the refinement

Finally, we introduce a practical example of these three aspects.

1. Identify the most general task

The most general task is the task that involves all the user needs that the Web application must support in a general way. This task can be identified from the Statement of Purpose.

We have seen in the previous section that one of the three aspects that the SP should describe is the purpose for which the Web application is developed. It is described in terms of which general service the system must support. Analyzing this sentence in detail we can extract from it the main activity that users can perform throughout this general service. This activity can be considered as the most general task which involves all the user needs. For instance, the purpose of the Amazon example “*is to provide support for the on-line purchase of products*”. We can extract from this sentence that the main activity that users should perform with the Web application is to purchase products. All needs that users present are related to the activity of purchase products. Thus, the most general task in the Amazon example is “*purchase products*”.

4. A Task-based Requirements Model for Specifying Web Applications

Then, we already know which the main task that users can perform is. However, it is too abstract and ambiguous in order to constitute a precise and complete requirements specification. We need to indicate more details about the task itself and about how users must perform it. For instance, following with the Amazon example, we need to detail aspects such as how product are purchased, who can purchase these products, how products are putting on sale in order to be purchased, and so on. We explain next how this can be done.

2. Refining the most general task into more specific ones

In order to refine the most general task into more specific tasks we propose to perform a hierarchical task analysis (HTA) [Sheperd 2001] from the most general task (see Section 4.1.2). Next we explain different aspects about how the HTA analysis must be performed.

HTA Aspect 1: Representation of Tasks. HTA proposes to define tasks by expressing the activities that they represent in an imperative way. For instance, we have described above the most general task in the Amazon example as “purchase products” (an imperative way of represent the activity of purchase of products). This form of description is versatile and can be used to describe tasks associated to broader activities such as “manage E-commerce” or tasks associated to more focused activities such as “query product information”. This aspect facilitates the description of tasks at different level of detail providing us with a valuable technique to refine the most general task and then to provide detailed information about it.

Historically, HTA techniques have used tabular or graphical representations to specify tasks. On the one hand, tasks are textually specified by means of tables when a tabular representation is used. Each task is represented by a row of the table. Task refinements are indicated by using for instance a specific task numeration (e.g. the Task 1.1 is a subtask of the Task 1). On the other hand, tasks are specified by a tree whose nodes represent tasks and whose branches represent task refinements when a graphical representation is used. Tasks in a level of the tree constitute the subtasks of the task in the upper level to which they are directly connected.

We propose the use of a graphical representation because we consider it to be more intuitive and easier to manage than tabular descriptions which become overloaded and ambiguous in the case of large systems. Currently, there are several notations that allow us to graphical represent tasks. For instance, we can find approaches such as CTT [Paternò 1997], GTA [Veer *et al.* 1996] or TKS [Johnson *et al.* 1992]. These approaches are usually focused on design activities and they provide a myriad of notations to represent tasks of different types as well as the refinement of them. They are very powerful tools to achieve the purpose of facilitating the design of software. However, RE activities require to use simpler notations in order to facilitate customers to understand the specified requirements.

4. A Task-based Requirements Model for Specifying Web Applications

Thus, we use a very simple notation based on a task taxonomy which is defined from ellipses and lines between them. On the one hand, each ellipse represents a task. Text inside the ellipse indicates the name of the task (see Figure 4.2A). The root of the taxonomy is defined by only one ellipse which constitutes the most general task. Ellipses in upper levels in the taxonomy indicate more general tasks. Ellipses in lower levels indicate more specific tasks. Ellipses in the leave level represent the most specific tasks. Levels in a task taxonomy are illustrated in Figure 4.2B.

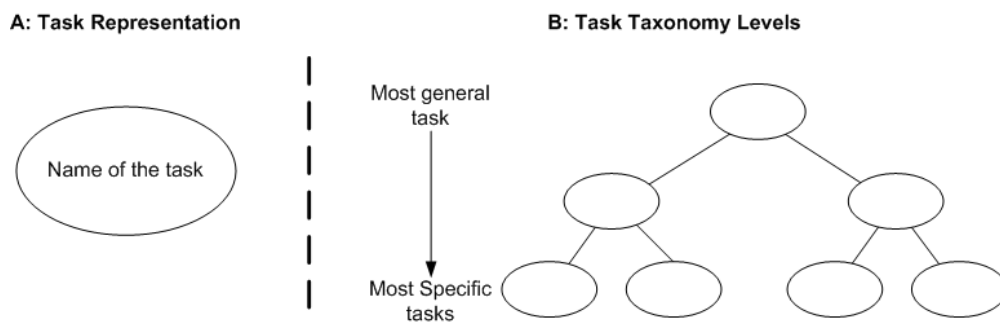


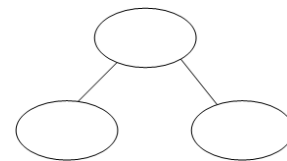
Figure 4.2 Graphical notations for tasks

The selection of a notation for representing tasks is not enough to correctly perform a decomposition of tasks. We must also select a criterion that guides us during the process of decomposition. Next, we explain it in detail.

HTA Aspect 2: Criterion for decomposing tasks. In order to facilitate the identification of specific tasks from more general ones we propose two kinds of task decomposition (or task refinement): structural refinement and temporal refinement. Task refinements are represented by the lines that connect tasks in the task taxonomy. Each line (refinement) can only connect a task A to a task B if B is in the level immediately lower to the level of A. The task connected to the line source (A, the task in the upper level) constitutes a parent task; the task connected to the line target (B, the task in the lower level) constitutes a subtask. Next we explain each type of task refinement in detail.

Structural refinement

Depicted by solid lines



This refinement decomposes a task (the parent task) into a set of simpler tasks (subtasks). Each of these subtasks can be performed independently to each other (there is no relationship among the performance of one subtask and the rest of subtasks). It is depicted by solid lines.

4. A Task-based Requirements Model for Specifying Web Applications

This refinement should be used when the activity associated to a task involves a set of independent user needs. By independent user needs we mean needs that can be separately supported by the Web application. For instance, the most general task in the Amazon example involves user needs that are all related to the purchase of products. However, if we analyze the SP (from which the most general task is defined) we can see how, according to the general description of functionality (third aspect included in the SP, see Section 4.2.1,) the Web application must support two types of user needs in order to support the purchase of products:

- Those related to the process of purchase products: [...] *the user must be able to consult information about the products, add them to a shopping cart and send purchase orders.*
- Those related to management activities: [...] *tools for the information management must be provided.*

These two types of needs can be handled separately. In fact, it is probably that they represent needs of two different types of users (products may be purchased by clients while management activities may be done by administrators). In these situations, we can group the different types of needs defining a subtask for each group. User needs are grouped by taking into account semantic and/or functional relationships. In the Amazon example, needs are grouped considering the way in which information is handled: in the case of needs related to the process of product purchase, information is shown to users; in the case of needs related to management activities, information is managed by users. Each created group gathers a set of needs that present a great level of affinity according to the selected criterion.

Thus, the task *purchase products* can be decomposed into two subtasks. Analyzing the type of user needs involved in this task, the two sub-tasks can be described as *purchase products* (which involve the needs related with the process of purchase products) and *manage information* (which involves the needs related to manage activities). However, the name of the first subtask is the same as the name of the parent task which can be confusing. So, we rename this subtask as *buy products*. This decomposition is shown in Figure 4.3

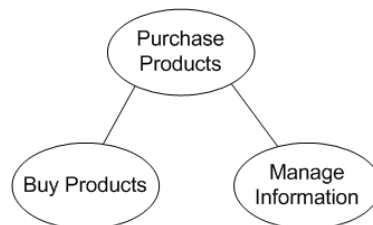
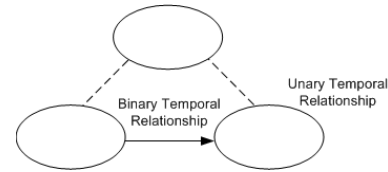


Figure 4.3 Example of structural refinement

4. A Task-based Requirements Model for Specifying Web Applications

Temporal refinement

Depicted by dashed lines



This refinement also decomposes a task (the parent task) into a set of simpler tasks (subtasks). However, this refinement provides constraints for ordering subtasks according to the parent task logic. This is also known in HTA as a plan [Shepherd 2001]. It is depicted by dashed lines.

This refinement should be used when the user needs that are involved in a task constitute a whole activity that can be partitioned in several coordinated parts. For instance, in the Amazon example, the activity associated to the task *buy products* (the whole activity) can be partitioned into two parts: (1) users collect the products into the shopping cart and (2) users checkout. In this context, the whole activity constitutes a parent task and each part constitutes a subtask.

In order to complete the whole activity all the parts in which it is partitioned must be completed. This means that to perform the parent task (and then to correctly support the user needs that it involves) all its subtasks must be performed. For instance, considering again the buy of products, users cannot buy products without collecting them to the shopping cart. In the same way, users cannot buy products without checkout. In order to buy products users must both collect products and checkout.

Another aspect to be considered in this kind of refinement is the way in which subtasks are performed. We have explained above that the whole activity that represents the parent task is decomposed into *coordinated* parts that represent subtasks. By coordinated parts we mean subtasks that cannot be performed in an arbitrary way. We need a plan that indicates the coordinated way in which subtasks must be performed. For instance, to buy products users must *first* collect them to the shopping cart and *next* check out. Users cannot buy products by first checking out and then collecting the products.

In order to define this coordination among subtasks (this plan) we propose the use of temporal constraints. We make use of the temporal relationships introduced by the CTT approach (ConcurTaskTree) [Paternò 1997]. CTT introduces a task model whose main purpose is to support the design of industrial applications. In this context, the temporal relationships introduced by CTT are defined for design purposes and they are not all suitable for the goal of this thesis, to capture requirements of Web applications. Next, we present the temporal relationships that are used in this thesis. We classify them into two types: (1) binary relationships, which are those relationships that relate two tasks and (2) unary relationships, which are those relationships that are applied over only one task. These relationships are the following:

4. A Task-based Requirements Model for Specifying Web Applications

Binary Temporal Relationships:

- $T1 \gg T2$, *Enabling*: This relationship indicates that the second task T2 must be always performed after the first task T1.
- $T1 []\gg T2$, *Enabling with information passing*: This relationship indicates also that T2 must be performed always after T1. However, in this case when T1 finishes it provides some data for T2.
- $T1 |> T2$, *Suspend-Resume*: This relationship indicates that during T1 is being performed it is interrupted in order to perform T2. When T2 terminates T1 can be resumed from the state reached before. T2 can interrupt T1 several times. Each interruption means the completed performance of T2.
- $T1 |= T2$, *Task independence*: This relationship indicates that tasks can be performed in any order, but when one starts then it has to finish before the other one can start.
- $T1 [> T2$, *Disabling*: This relationship indicates that the T1 (usually an iterative task) is completely interrupted by T2. T1 is not resumed again.

Unary Temporal Relationships:

- $T1^*$, *iterative task*: the task can be completed several times.
- $T1(n)$, *finite iteration*: how many times the task will be performed is specified.
- $[T1]$, *optional task*: its performance is not mandatory.

We represent temporal relationships by attaching them to an arrow that connects the two related tasks. Figure 4.4 shows the temporal decomposition introduced above: The task *buy products* is decomposed by means of a temporal refinement into the tasks *collect products* and *checkout*. The temporal relationship that is used is *enabling with information passing* ($[]\gg$) that indicate that the task *checkout* must be performed after the task *collect products*. The information that is passed is the collected products.

Temporal relationships can be only defined between sister tasks, that is, between subtasks of a same parent task (of course if the parent task has been decomposed by means of a temporal refinement). For instance, considering the subtask *collect products*, we can define a temporal constraints between it and its sister subtasks *checkout* and vice versa. No other task than *checkout* can be related to *collect products*. No other task than *collect products* can be related to *checkout*.

4. A Task-based Requirements Model for Specifying Web Applications

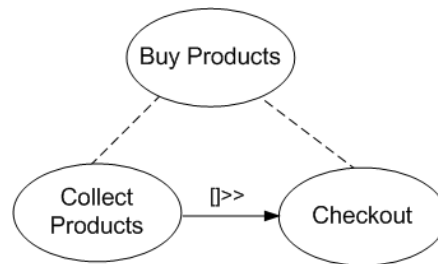


Figure 4.4 Example of temporal refinement

HTA Aspect 3: Information for decomposing tasks. The SP provides valuable information in order to identify a set of initial tasks. This set of tasks can be taken as base to perform the task decomposition. For instance, we have seen above how analyzing the description of functionality that is included in the SP of the Amazon example we have decomposed the most general task into two subtasks (one that supports the purchase of products and another that supports management activities).

However, the SP is not enough to completely perform the task decomposition. For instance, the SP does not indicate how products must be purchase or which management activities must be supported. In order to collect this information, analysts must access other sources such us stakeholders, documents used by an organization, previous applications, usual activities performed by employees and so on. To do this, proper elicitation requirements techniques must be used. However, they are out of the scope of this thesis.

3. Deciding when stopping the refinement

One of the most difficult aspects in hierarchical task analysis is the following: when we know that the decomposition of tasks is finished.

This problem was already discussed at the beginnings of HTA [Annett & Duncan 1967]. HTA was initially proposed to be a general method for examining work. Annet and Duncan suggested a stopping rule known as the PxC rule: analysts should finish the decomposition of a task when the probability of failure (of performing an inadequate decomposition) (P) multiplied (x) by the cost of failure (C) surpasses a specific predefined level of acceptance. In this case, analysts should estimate P, should estimate C and should establish the predefined level of acceptance. This is not always easy, as admit the authors Annet and Duncan.

Other more recent approaches such as [Shepherd 1993] or [Omerod et al. 2003] propose a rule based on the analysis of goals. They propose to finish the refinement when the goal that must be achieved by a task is of lowest level. They introduce different types of goals of lowest level in order to facilitate their identification. For instance, they explain that a goal of low level can be a goal that only implies an action

4. A Task-based Requirements Model for Specifying Web Applications

which changes the state of the system or those that only implies an observation of the state of the system.

In this thesis, we are inspired by this last approximation. However, we want to provide a more practical rule for the purpose of identifying the set of the tasks that represent the user needs. To do this, we must analyze the definition of task that is provided above. We consider a task to be *an activity that users need to perform by interacting with the web application in order to achieve a specific goal*. According to this definition, a task represents an activity that users must perform together with the system. Thus, we stop decomposing tasks when a subtask constitutes an atomic action. We consider atomic actions to be either single actions of the user or single actions of the system. A single action of the user can be for instance the selection of information or the activation of an operation. A single action of the system can be for instance the inquiry of the state of the system or the modification of it. These actions are studied in the next section.

Thus, the refinement stopping rule that we propose in thesis is:

A task must not be decomposed when atomic actions are obtained in its decomposition.

We call *elementary tasks* to those tasks that should not be decomposed.

4. A practical example

Next, we introduce the task taxonomy of the Amazon example. It is shown in Figure 4.5 and it is described as follows:

- From the statement of purpose we extract the most general task that is *Purchase Products*.
- The task *Purchase Products* is decomposed by means of a structural refinement (solid line) into two tasks: (1) *Buy Products* which involves the needs related to the process of buy products and (2) *Manage Information* which involves the needs related to management activities.
- The task *Buy Products* is decomposed by means of a temporal refinement (dashed line) into *Collect Products* and *Checkout*. The relation between them is *enabling with information exchange*. Thus, the products should be collected into the shopping cart before checkout. The information that needs to be exchanged is the collected products.
- In order to collect products, users add them to the shopping cart. During this process, users can inspect the shopping cart when they consider opportune.

4. A Task-based Requirements Model for Specifying Web Applications

Thus, *Collect Products* is decomposed into *Add Product to Shopping Cart* and *Inspect Shopping Cart*. The relation between the two tasks is *suspend-resume*, which indicates that *Add Product to Shopping Cart* may be interrupted at any point by *Inspect Shopping Cart* (this is not mandatory, see how the second task is defined as an optional task). After each interruption, the task *Add Product to Shopping Cart* is resumed.

- *Add Product to Shopping Cart* is an iterative task. Then, it can be performed so many times the user needs. It is decomposed by means of a structural refinement into the tasks *Add CD*, *Add Software*, and *Add Book*. Then, when the user is adding a product to the shopping cart he/she is adding a CD, a Software product or a Book.
- The task *Manage Information* is decomposed by means of a structural refinement into *Manage Products* and *Manage Customers*. Users can manage information about either products or customers.
- Finally, the task *Manage Products* is decomposed by means of a structural refinement into *Manage CDs*, *Manage Software* and *Manage Books*. Users can manage information about CDs, software or books.

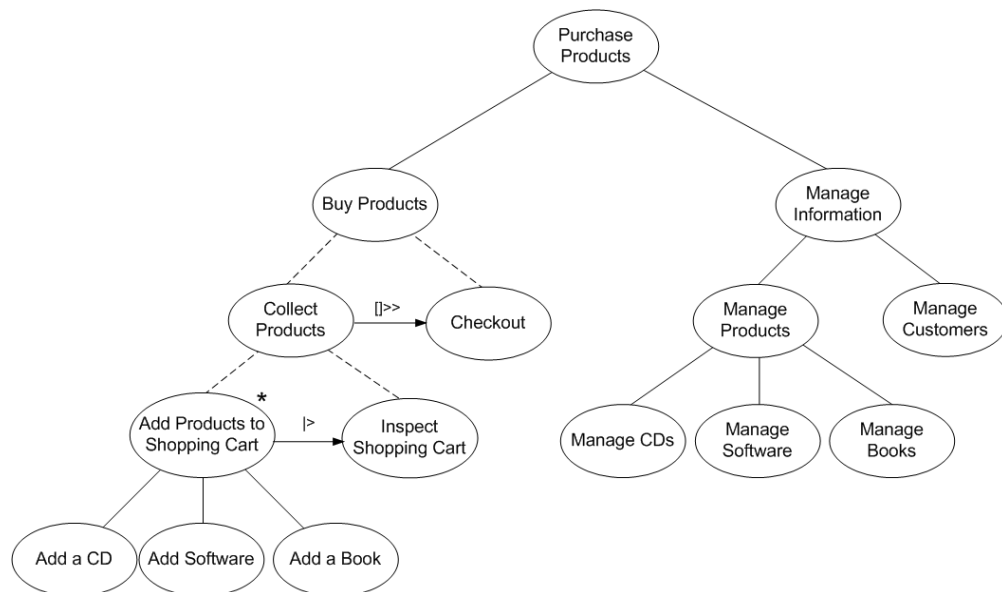


Figure 4.5 The task taxonomy of the Amazon Example

4. A Task-based Requirements Model for Specifying Web Applications

Activity 2: Description of (Elementary) Tasks

Next, we introduce a technique to describe elementary tasks. We limit the use of this technique to the description of elementary tasks since these tasks constitute the lowest level of the taxonomy. Then, contrary to non-elementary tasks, which are described throughout their subtasks, there is no description in the taxonomy for elementary tasks.

We describe elementary tasks by following two steps:

1. Characterization of Elementary Tasks
2. Description of Elementary Task Performance

1. Characterization of Elementary Tasks

The characterization of an elementary task performance must indicate information related to aspects such as:

- *why* the task must be performed
- *who* can perform the task
- *how often* the task is performed
- *which restrictions* must be considered in its performance

To do this characterization, the following information is proposed to be defined for each elementary task:

- *Goal*: Mandatory information. It indicates the objective that must be reached after performing the task. Before defining the actions of the user and the system that describe each elementary task, we must clearly indicate the goal that these actions must achieve. This aspect facilitates the description of elementary tasks and it can be also used to validate requirements in further stages. If the actions that describe an elementary task achieve the predefined goal then the user needs that are involved in this task are correctly captured.
- *Users*: Mandatory information. A very important aspect to be considered in the performance of a task is the users that can perform it. Web applications are accessed by users with different profiles (visitors, customer, administrator, etc). Indicating the type of users that can perform each elementary task we provide a mechanism to manage these profiles at the requirements level. Furthermore, it can be also used to provide adapted solutions to problems of specific users. The

4. A Task-based Requirements Model for Specifying Web Applications

adaptation topic is out of the scope of this thesis. However, a preliminary work about this topic can be found in [Rojas *et al.* 2006].

- *Frequency*: Optional information. It describes the relative frequency of occurrence of the task. This is a very useful aspect in the description of a task that may affect the Web application development. For instance, if the frequency of a task A is much greater than the frequency of a task B, the Web application should be implemented by following a strategy that optimizes the completion of task A.
- *Additional Constraints*: Optional information. Further conditions that must be fulfilled in order to complete a task. These conditions can be related to performance, security, availability, etc.

In order to describe this information we associate a template to each elementary task. Figure 4.6 shows the template associated to the elementary task *Add CD* (see Figure 4.5). According to this figure, the goal of the task is for the user to add a CD to the shopping cart; the task can be completed by visitors and customers. As an estimate, the task is going to be completed 100 times per hour. Finally, no additional constraints are defined.

<u>Task:</u>	Add CD
<u>Goals:</u>	The user adds a CD to his/her Shopping Cart
<u>Users:</u>	Visitor, Customer
<u>Frequency:</u>	100 times per hour
<u>Additional Constraints:</u>	_____

Figure 4.6 Information related to the performance of *Add CD*

2. Description of Elementary Task Performances

Templates for the task characterization allow us to indicate aspects related to why, by who or how often elementary tasks are performed. Next we introduce a technique to describe *how* elementary tasks must be performed.

The performance of tasks is traditionally described, in the specification of software requirements, from the set of actions that both the system and the user perform during the task [Lauesen 2003]. These descriptions are very suitable to capture functional

4. A Task-based Requirements Model for Specifying Web Applications

requirements. However, when they are used to capture Web application requirements they fail in the specification of navigational requirements.

As we have already explained, the main responsible of the great encouragement of Navigation in Web applications is the focus of this type of software, which is mainly centered on communication, i.e. on providing users with information. In this sense, the way in which users interact with Web applications substantially changes from the way in which users interact with traditional software. In the case of traditional software, users interact with the system to activate some functionality in order to obtain specific results. In the case of Web applications, users can moreover interact with the system in order to just browse information. We think that providing a mechanism to capture this type of interaction at the requirements level (how users browse information) we are providing a mechanism to properly capture the navigational requirements of Web applications.

In this context, we extend traditional description of tasks (based on a sequence of actions of the system and the user) by introducing explicit information about each time that the user and the system interact to each other. In each of these interactions, we indicate the information that is browsed by users (navigational requirements). To represent these interactions at the requirement level we introduce the concept of *interaction point (IP)*.

Thus, the technique that we propose allows us to describe the performance of elementary tasks by using three elements: system actions, user actions, and interaction points. The performance of an elementary task is considered to be a process where the system carries out several *system actions*, sometimes delaying them in order to interact with the user by means of *interaction points*. In these interaction points, users access information and have the possibility of performing several *user actions* with them. In order to perform this type of descriptions we propose the use of UML 2.0 *activity diagrams* [UML]. We have chosen this diagrammatic description instead of a traditional textual description because as [Vilian et al. 2000a] states, textual descriptions present a lack of precision and conciseness when they are used in a validation process with users.

This technique constitutes one of the most important contributions of this thesis. Its main property relies on combining mechanisms to describe the interaction between the user and the system (aspect which has been traditionally handled in the field of HCI [Grechenig & Tscheligi 1993]) with mechanisms to capture transactional requirements (aspect which has been traditionally handled in the field of RE [Hofman 2000]). The result of this combination is a technique for describing tasks considering not only transactional requirements but also navigational ones.

As a first contact with this technique, Figure 4.7 shows part of the description of the elementary task Add CD (this task is completely presented further). In particular, this figure describes how users access a CD description. System actions are represented

4. A Task-based Requirements Model for Specifying Web Applications

by dashed nodes; IPs are represented by solid nodes; user actions are represented by arcs. According to Figure 4.7, users start by accessing a list of CDs. From this list, users can either select one and then they obtain the description of the selected CD, or activate a system action that performs a search of CDs. The criterion of this search is a specific artist which is introduced by the user. From this search, the system provides users with a description of a CD (if only one CD is found) or with a list of CDs (if a list of CDs is found).

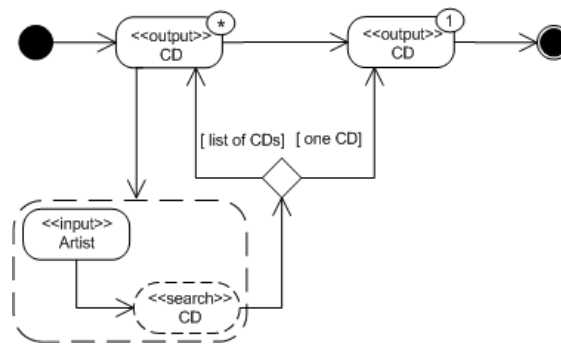


Figure 4.7 A first example of a task performance description

Next, we explain this technique in detail by explaining how system actions, interaction points and user actions are described by means of activity diagrams.

System Actions

System actions describe the actions that the system must perform in order to achieve the goal of an elementary task. We propose two types of system actions depending on whether or not they modify the system state. We said that a system action changes the system state when its performance produces a concrete reaction (change of state) in the system. For example, when a product is added to the shopping cart in the Amazon example, the corresponding modification in the shopping cart should be recorded (and then, the system state changes).

Thus, we distinguish the following types of system actions:

- *Functionality Execution*: This type involves those actions of the system that change its state. For instance, the introduced above action of adding a product to the shopping cart.
- *Information Search*: This type involves those actions of the system that only query its state. An example of this type of actions is the search introduced above, where the system search the list of CDs that belong to a specific artist.

4. A Task-based Requirements Model for Specifying Web Applications

In this case, after performing the system action there is no reaction in the system. The system state is not changed.

Representation Aspects: System actions are represented by nodes of the activity diagram. In order to graphically distinguish these nodes from the nodes that represent interaction points (further explained) we depict them with dashed lines. Furthermore, nodes are stereotyped with the *Function* or the *Search* keyword in order to indicate their type: functionality execution or information search respectively. Figure 4.8 shows the graphical representation of system actions.

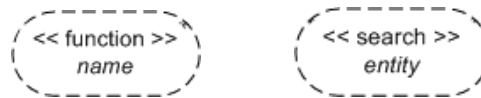


Figure 4.8 System Action representations

For each functionality execution action we must indicate an action name which properly describes the change that it produces in the system state. For instance, we can name an action which adds products to the shopping cart as *add_product_to_shopping_cart* (or just *add_product* if the shopping cart is inferred).

As far as information search actions, they do not change the system state, they just query it. We consider that this type of system actions query the system state in order to search information about an entity. By entity we mean any object of the real world which is implicated in the performance of a task and which the system must know in order to correctly perform the task (e.g. client, product, invoice, etc). So, for each information search system action we must indicate the entity that they are querying. For instance, a system action which searches the list of CDs that belong to a specific artist is associated to the entity CD.

Interaction Points (IP)

As stated above, we introduce the concept of interaction point in order to represent each time that the system interacts with the user during a task performance.

We propose two kinds of interaction between the system and the user depending on the objective for which it takes place. We consider that an interaction can take place in order to allow the system provides information to the user or vice versa, in order to allow the user to introduce information into the system. To capture these two ways of interaction we propose two kinds of IPs:

- *Output IP:* It captures an interaction in which the system provides the user with information. Access to operations can be also provided.

4. A Task-based Requirements Model for Specifying Web Applications

The information is related to a specific entity¹⁸ of the system. For instance, during the task of adding CDs to the shopping cart, users should be able to access a description of the CD that they want to add. The interaction in which the system provides users with this description is represented as an output IP that provides information related to the entity CD.

As far as the access to operations, two types are distinguished:

- Access to operations related to the IP entity. In the same way that an IP provides information that allows users to study an entity, it can also provide operations that allow users to manipulate the entity. For instance, in the example introduced above, when users access the CD description they should be able to access an operation for adding it to the shopping cart. This operation is related to the IP entity (CD) and it allows users to manipulate CDs (by adding them to the shopping cart).
 - Access to operations related to the logic of a task. An Output IP represents a step of a task performance in which the system provides users with specific information. It is possible that, according to the task logic, the next step of a task be a system action which, although it is not related to the IP entity, users must be able to access from the IP in order to correctly perform the task. For instance, considering the Amazon example, when users are inquiring the different items that are in the shopping cart, they should be able to identify themselves in order to checkout. This identification is represented as a system action. This system action is not related to the entity associated to the IP that provides information about the shopping cart (Item), however, users must be able to access it from this IP in order to accomplish with the task logic.
- *Input IP*: It captures an interaction in which the system requests the user to introduce information of an entity. The system uses this information to correctly perform a specific action. For instance, during the checkout, the system needs specific client information in order to create the purchase order. The interaction in which this information is introduced is represented by an input IP. The introduced information is related to the entity client.

Representation Aspects: Interaction Points are represented by nodes of the activity diagram. These nodes are depicted with solid lines. Furthermore, nodes are stereotyped with the *Output* or the *Input* keyword in order to indicate their type. In

¹⁸ Any object of the real world that belongs to the system domain (e.g. client, product, invoice, etc)

4. A Task-based Requirements Model for Specifying Web Applications

this case, depending on the type of IP we must also indicate the following characteristics:

- For the *Output IPs* (see left side of Figure 4.9): The number of information instances¹⁹ that the IP includes (cardinality) is depicted as a small circle in the top right side of the primitive. We can indicate either a fixed cardinality (a specific number of instances are provided) which is indicated with a natural number or a variable cardinality (a no predefined number of instances is provided) which is indicated with the symbol '*'.
- For the *Input IPs* (see right side of Figure 4.9): We know that this type of IPs represent the interactions in which the system requests the user to introduce information of an entity. We have explained above that the system uses this information to correctly perform a specific action. In this context, Input IPs are defined only when the system requires user information in order to perform a specific action. Thus, these IPs exclusively depend on the performance of a specific system action. They cannot be considered to be one more step in the description of a task performance. In order to graphically capture this aspect we encapsulate both elements (input IP and system action) in a dashed square.

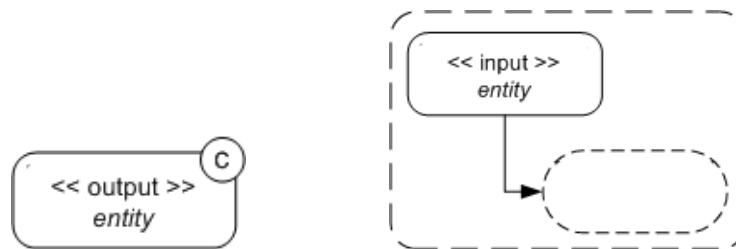


Figure 4.9 IP representations

User actions

We have seen how IPs are used to capture each time that the system and users interact to each other. These interactions allow user to perform several actions. These actions depend on the type of IP. They are the following:

- Considering the Output IPs, users can perform actions with both the information and the operations that IPs provides:
 - User can *select information*: Output IPs provide information about an entity and then users can select part of this information in order to study it.

¹⁹ Given a system entity (e.g. client), an information instance is considered to be the set of data related to each element of this entity (Name: Joseph, Surname: Elmer, Telephone Number: 9658789).

4. A Task-based Requirements Model for Specifying Web Applications

For instance, let's consider again the Output IP which provides the user with information about a CD. This information may include the name of the artist. Thus, the user may want to select the name of the artist in order to access more detailed information about this artist. The result of this action is that the system provides the user with new information (which is related to the user selection).

- The user can *activate an operation*: Output IPs also provides access to operations and then users can activate these operations. For instance, the above introduced Output IP provides access to an operation that adds a CD to the shopping cart. The result of this user action is that the system performs an action (this one that support the operation activated by the user).
- Considering the Input IPs, the only action that users can perform is the *input of information*. The result of this action is that the system obtains required information to perform an action.

Representation Aspects: Users Actions are represented by arcs in the activity diagram. In this case, only minor extensions are introduced over the representation of an UML activity diagram arc in order to capture the different types of user actions. We can derive the type of user action that represents an arc directly from the type of the nodes (IP or system action) that it connects.

We have introduced above the different actions that users can perform during the performance of a task. These actions are performed from the interactions with the system. These interactions are captured by Interactions Points. In this context, an arc in the activity diagram represents a user action if its source node is an IP. The type of user action can be derived from the target node of the arc. Next, we indicate which connection of nodes allows us to derive each type of system action.

- *Selection of Information*. This action is captured by those arcs that connect two Output IPs. The system provides the user with information by means of the first Output IP (source of the arc). The user selects part of this information and then the system provides additional information related to the selection of the user by means of the second Output IP (target of the arc).

Example: Figure 4.10 shows two Output IPs that are connected by an arc. The first Output IP provides the user with a list (*, cardinality variable) of CDs. The second Output IP provides the user with information about a specific CD. The arc represents the user action of selecting a CD from the list in order to obtain a detailed description of it.

4. A Task-based Requirements Model for Specifying Web Applications

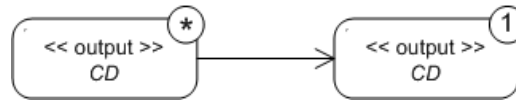


Figure 4.10 Example of user action: selection of information

Double arrowed arc: If the arc is doubly arrowed it indicates that after the user finishes of studying the information provided by the second IP the user must return to the first IP in order to continue with the task performance.

These situations represent those moments during a task performance in which the user may access complementary information. This complementary information is provided by the Output IP defined as target of the arc. It complements the information provided by the Output IP defined as source of the arc. It is not mandatory for the user to access this information in order to perform the task. However, it can help the user to take decisions about the information and operations provided in the source Output IP.

For instance, let's consider a moment during a task in which users access the description of a CD and then users have the possibility of adding the CD to the shopping cart. This moment is represented by an Output IP associated to the entity CD (see Figure 4.11). This IP is connected to the system action *Add_CD* which indicates that for adding the CD to the shopping cart users just need to activate this operation (this is explained next). In this case, we want to provide users with the possibility of inquiry complementary information about the artist of the CD. However, we want to indicate that the access to the artist information is optional and users must return to the CD description in order to achieve the task goal (add the CD to the shopping cart). We can capture this aspect by means of an Output IP which both provides information about an artist and is connected to the Output IP that provides the CD description by means of a double arrowed arc. An example of this is shown in Figure 4.11. According to this figure users have the possibility of inquiring information about the artist of a CD before adding it to the shopping cart.

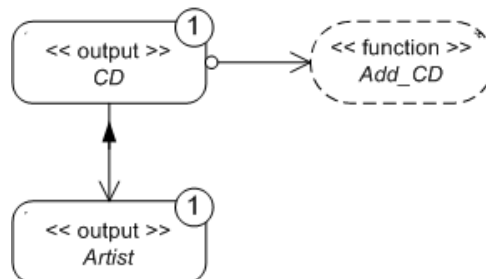


Figure 4.11 Example of double arrowed arc

4. A Task-based Requirements Model for Specifying Web Applications

- *Activation of Operations.* This action is captured by those arcs which connect an Output IP (arc's source) with a system action (arc's target). The system provides users with access to one or more operations by means of the Output IP. The user selects the operation that is represented by the target node of the arc (the system action). Then, the system performs the action.

As explained above, Output IPs provide users with two types of operation access: access to operations related to the IP entity and access to operation required to correctly satisfy the task logic. Thus, two types of operation activation are distinguished as far as the user actions: (1) activation of an operation related to the entity and (2) activation of an operation related to the task logic. To distinguish them, a minor extension in the representation of arcs is introduced. This minor extension consists in attaching a small circle in the arc source when the activation of an operation belongs to the first type.

Example: An example of an operation activation related to the IP entity has been already introduced in Figure 4.11. This figure shows an Output IP that provides information about a CD (a description of a CD). This Output IP is connected to the system action *Add_CD* by an arc (with a small circle in its source). This arc represents the user action of activating the operation *Add_CD*. This operation is directly related to the IP entity (CD) because it allows users to manipulate the entity.

An example of the second type of activation of operation is shown in Figure 4.12. According to this figure, users can activate the operation of *Login* from an Output IP that provides users with information about the *Items* added to the shopping cart. In this case, the operation is not directly related to the IP entity (Item). Users must activate the operation in order to satisfy the task logic.

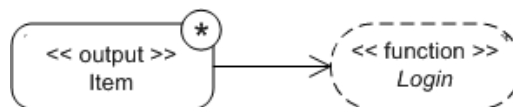


Figure 4.12 Example of operation activation

- *Introduction of information.* This action is captured by those arcs that connect an Input IP (arc's source) with a system action (arc's target). A system action requires that users introduce some data in order to be correctly performed. Users introduce this data by means of the Input IP and then the system performs the action.

Example: Figure 4.13 shows an Input IP and a system action that are connected by an arc. In this case, the arc represents the completion of the information input in order to allow the system to use it. According to this example, the

4. A Task-based Requirements Model for Specifying Web Applications

system must search information about CDs. This search must be performed according to a specific criterion. This criterion is an artist and it is introduced by users throughout the Input IP. Thus, once users have finished of introducing the criterion (an artist) the system uses it in order to search CDs (CDs of a specific artist).

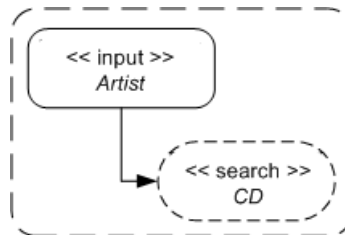


Figure 4.13 Example of user action: introduction of information to perform a system action

- *Activation of operation + Introduction of information.* As explained above, the system can provide users with access to one or more operations by means of Output IPs. The user action of activating an operation is represented by connecting the Output IP to the corresponding system action by means of an arc. However, if the system action requires user information that is introduced through an Input IP, the Output IP must be connected to the couple of nodes Input IP-System action. In these cases, users selects an operation (the system action) that requires user information, next the user introduces the required information by means of the Input IP, and finally the system performs the corresponding action by using the user information.

Example: According to Figure 4.14, the system provides the user with a list of CDs by means of an Output IP. From this list, the user can activate a search system action. This action searches the CDs of a specific artist. This artist, which is used as search criterion, is introduced by the user through the Input IP.

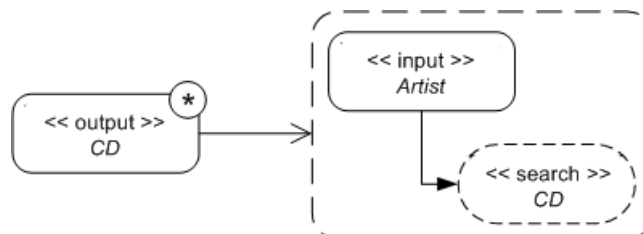


Figure 4.14 Example of user action: activation of operation + introduction of information

4. A Task-based Requirements Model for Specifying Web Applications

Additional Information

We have already explained how the different elements of a task performance description are represented by means of activity diagrams. However, this type of diagrams allows us to represent additional information. This additional information enriches the descriptions of task performances allowing us to obtain a powerful Web application requirements specification based on the concept of task. Next, the representation of this additional information is introduced.

- *System Action Sequence.* Arcs do not always represent user actions. If an arc connects two system actions it represents a sequence of system actions. In these cases, arcs only represent the order in which a set of system actions must be performed.

Example: Figure 4.15 introduces a sequence of system actions. According to this figure the system first adds a CD to the shopping cart and next it updates the product stock.

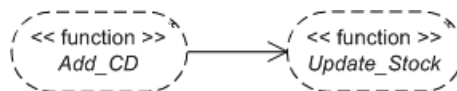


Figure 4.15 Example of sequence of system actions

- *Information Transfer.* If the source of an arc is a search system action the arc implicitly represent moreover an information transfer. Search system actions are those actions which inquiry the system state in order to search specific information. This search obtains a result and then this result must be properly handled. It can be handled in two ways:
 - It can be shown to users.
 - It can be passed to other system action which uses it.

In order to indicate this aspect, we connect the search system action with the element that must use the search result. For instance, Figure 4.16 shows the search system action presented above (Figures 4.13 and 4.14). This action searches the CDs of a specific artist. In this case, we have connected the action with an Output IP. This means that the results obtained by the search system action are shown to users through the Output IP.

4. A Task-based Requirements Model for Specifying Web Applications

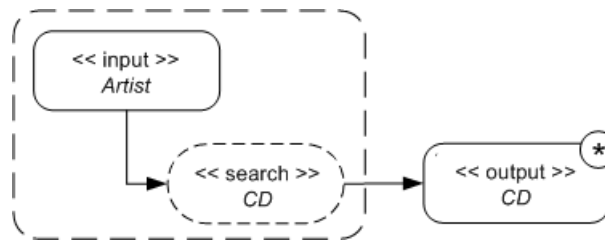



Figure 4.16 Example of transfer of information

- *Conditioned Flow.* Arcs are used to connect elements of a task performance description. These connections present different semantics depending on the elements that they connect. We have seen how an arc can represent a user action, a sequence of actions or an information transfer. However, in all these cases arcs represents a flow of steps in a task performance. If we consider each element of an arc connection as a step of the task performance, we must do first the step represented by the arc's source and next the step represented by the arc's target. This flow can be constrained with several conditions:
 - Conditions over information selected by the user. This condition can be defined in the case that the arc's source is an Output IP. This IP provides users with information that they can select. We can define a conditioned flow depending on the selection of the user.
 - Conditions over the results obtained by a search system action. We can define a different next step depending on the results obtained by a search system action.
 - Conditions over the success or failure of a system action. We can define a different next step of a system action depending on the success or failure of the action.

In order to represent this conditioned flow we use the symbol proposed in the UML activity diagrams for representing decisions (). Conditions are defined by depicting them between brackets. Conditions are expressed in a natural language in order to be easily understand by customers. Figure 4.17 shows an example of conditioned flow. This figure shows the search system action present above. This action searches the CDs of a specific artist. If the actions return a list of CDs the next step is an Output IP that shows this list to the user. If the action returns nothing the next step is an Output IP that provides the user with a list of artists.

4. A Task-based Requirements Model for Specifying Web Applications

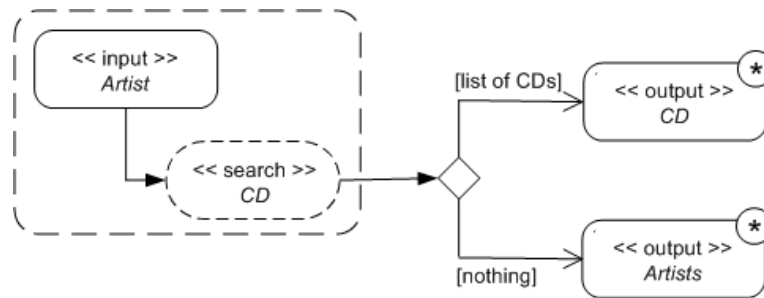


Figure 4.17 Example of conditioned flow

- *Initial and Final steps.* The initial node (●) and final node (●) of an UML activity diagram are used to indicate the first and last step in a task performance.

In order to indicate the initial step, we connect the node that represent this step to the initial node (●). The initial node can be connected to either a system action (Function or Search) or an Output IP. Thus, the first step of a task can be a system action or an Output IP. Input IPs cannot constitute the first step of a task by themselves. These IPs exclusively depend on a system action. As we have explained above, they are used to allow users to introduce the information that the system requires to perform a system action. Then, Input IPs define the first step of a task only in the case that its associated system action constitutes the first step of the task.

Depending on the type of the node that represents the initial step, a task starts in a different way:

1. If the node is a system action the task starts without an active participation of users. The system does not require the participation of users to start the task. The system performs the system action and then continues with the next step according to the task performance description.
2. If the node is an Output IP, the system provides the user with information in the first step of the task. Thus, in the first step of the tasks users analyze information. The task starts with an active participation of users. Furthermore, how the task continues depends on the action that users perform with the provided information.

Figure 4.18 shows an example of initial step. According to this figure, the task starts by providing users with a list of music categories.

4. A Task-based Requirements Model for Specifying Web Applications

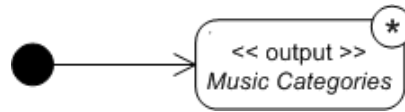


Figure 4.18 Example of initial step

In order to indicate the final step, we connect the node that represents this step to the final node (●). Only function system actions or Output IPs can be connected to the final node. Input IPs cannot be the final step because of the same reason explained above. Search system actions cannot be defined as the last step of a task because these actions are used by the system to query information. Once the information is obtained, it must be handled in some way (by showing it to users or by passing it to another system action). To do this, these actions need to be connected to either an Output IP or other system action. These actions cannot be connected to the final node.

Depending on the type of the node that represents the last step, a task finishes in a different way:

1. If the node is a Function system action the task immediately finishes when the system performs the action. The user does not explicitly decide to finish the task.
2. If the node is an Output IP the task ends when users finish of analysing the information provided in this IP. In this case, users must explicitly decide when the task is finished (when users finish of analysing the information).

Figure 4.19 shows an example of final step. According to this figure, the task is finished with the system performing the action *Update_Stock*.



Figure 4.19 Example of final step

Examples of Task Performance Descriptions

Next, we introduce some representative examples of task performance descriptions. In particular, we show the description of the following elementary tasks: Add CD, Inspect Shopping Cart and Checkout.

4. A Task-based Requirements Model for Specifying Web Applications

Example 1: Elementary task *Add CD*. The *Add CD* elementary task is described in Figure 4.20 (the shaded numbers are not part of the notation):

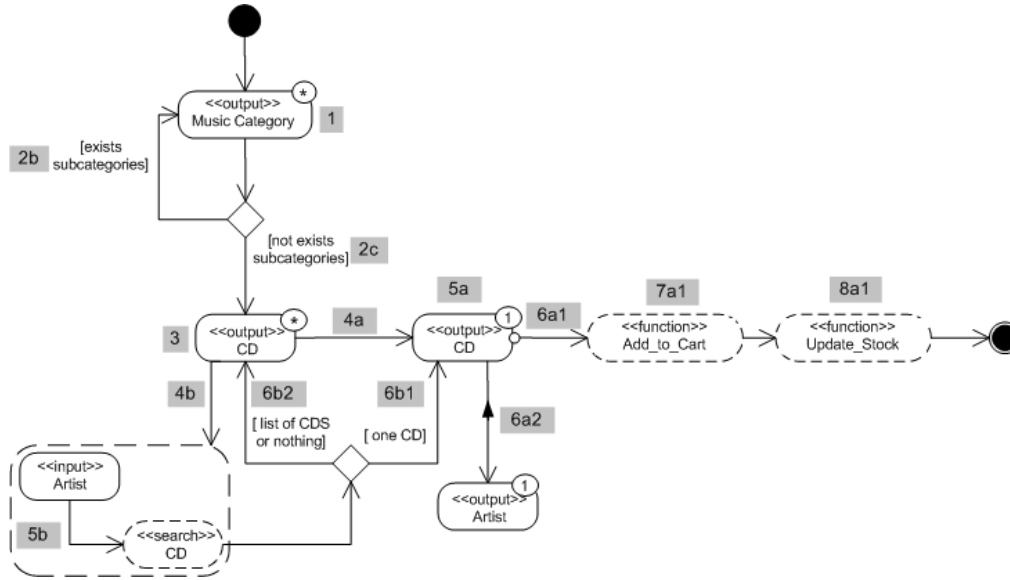


Figure 4.20 *Add CD* Elementary Task.

According to the description in Figure 4.20:

- The elementary task *Add CD* starts with an *Output* IP where the system provides the user with a list (cardinality *) of music categories (1).
- From this list, the user can select a category (2a and 2b). If the category has subcategories, the system provides the user with a list of (sub) categories (2b).
- If the selected category does not have subcategories (2a) the system informs about the CDs of the selected category by means of an *Output* IP (3). The user can perform two actions from this IP:
 - A) Select a CD (4a), and then the system provides the user with a description of the selected CD (5a).
 - B) Activate a search operation (4b), and then the system performs a system action that searches for the CDs of an artist (5b). To do this, the user must introduce the artist by means of an *Input* IP. If the search returns only one CD, the system provides the user with its detailed description (6b1). Otherwise, the system provides the user with a set of CDs (6b2).
- Finally, when the user has obtained a CD description (5a) s/he can:

4. A Task-based Requirements Model for Specifying Web Applications

- A) Select the artist of the CD (6a2), and then the system provides the user with detailed information about the artist (7a2). In this case, it is mandatory that the user returns to the CD description in order to achieve the task goal (to add a CD to the shopping cart, see Figure 4.2).
- C) Activate the *Add_to_Cart* operation (6a1), which is an operation that allows users to manipulate IP entities (see small circle at the arc source). When users activate this operation the system performs an action that adds the selected CD to the shopping cart (7a1). Next, the system updates the stock (8a1) and finally the task finishes. In this case, there is no an explicitly decision of the user to finish the task. The task is implicitly finishes when the user activates the action *Add_to_Cart*. The system performs this action, and then the next one, and then the tasks is end.

Example 2: Elementary Task *Inspect Shopping Cart*. Figure 4.21 shows another example of a task description. This figure shows the description of the elementary task *Inspect Shopping Cart*.

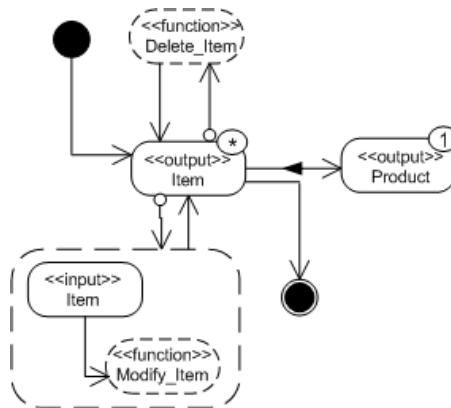


Figure 4.21 *Inspect Shopping Cart* Elementary Task.

According to Figure 4.21, the task *Inspect Shopping Cart* begins with an Output IP where the system provides users with the list of shopping cart items. From this list, users can modify an item or delete it. To modify an item, the system needs that users introduce the new data. To do this, an Input IP is defined. After the system has performed one of these actions it provides users with the updated list of items. Furthermore, users can select one item in order to access a description of the product associated to it. The access to this information is optional and users must return to the list of items in order to achieve the task goal (to manage the shopping cart). In this case, users must explicitly decide to finish the task. The task is finished when users finish of managing the items of the shopping cart.

4. A Task-based Requirements Model for Specifying Web Applications

Example 3: Elementary Task *Checkout*. According to Figure 4.22, the task *Checkout* begins with an Output IP where the system provides user with a description of its associated purchase order. From this description, users can: (1) create a new customer account, for which users need to introduce the information that is required to be a customer or (2) identify themselves as registered customers in order to process with the purchase. If users identify themselves correctly the system sent the purchase order; users need however to introduce credit card information previously. If the identification fails users access the purchase order description again.

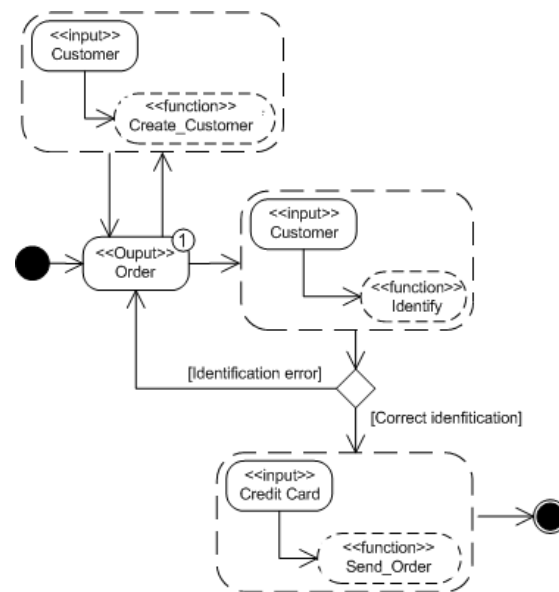


Figure 4.22 *Checkout* Elementary Task.

In this example, users can activate two system actions (*Create_Customer* and *Identify*) from the IP that provides the order description. These system actions must be activated from this IP in order to correctly perform the task. They are related to the logic of the task *Checkout*. They are not related to the entity *Order*. Finally, users do not decide explicitly to finish the task. The task is finished when the system performs the action of *Send_Order*.

4.2.3 System Data Description

Once user tasks have been identified and described, the next step consists in defining the data that the system must store about each identified entity (e.g. CD, Artist, Item, Order, etc). Furthermore, we also describe the data that is exchange in each IP that is defined in the task performance descriptions.

4. A Task-based Requirements Model for Specifying Web Applications

Definition of Informational Requirements

The information that must be stored in the system is defined by means of information templates. To do this, we have inspired by techniques such as NDT [Escalona 2004], the CRC Card [Wirfs-Brock *et al.* 1990] or the approach presented in [Durán *et al.* 1999].

We propose the definition of an information template (see Figure 4.23) for each entity identified in the description of an elementary task performance. We also propose to create information templates for entities that have not been identified in the description of task performance if they allow us to better structure the data that must be stored in the system.

In each template, we indicate an identifier, the entity, and a *specific data* section. In this section, we describe in detail the information that the system must store about the entity. To do this, a list of features about the entity is defined. For each feature, we specify the following fields:

- *Name*: The name of the feature.
- *Description*: A brief description of the feature.
- *Nature*: This field expresses the specific type of the feature in an abstract way, without giving design details. There are natures of two kinds:
 - (1) *Simple nature*: This nature represents a simple type. We use predefined values such as *string*, *number*, *text* and so on in order to represent this type. A simple nature can also be defined as a *list* of elements. In this case, we must indicate the type of the elements that are included in the list. This type is defined by one of the predefined values introduced above.
 - (2) *Complex nature*: This nature represents a type defined from another entity. We use the identifier of an information template to represent this type. This means that the nature of this feature is defined by the entity associated to this information template. We can also define a complex nature as a *list* of elements. In this case, the type of the elements is defined by indicating the identifier of an information template.

Figure 4.23 shows the information template associated to the entity CD (identified in the description of the elementary task *Add CD*, see Figure 4.20). According to this template, the information that the system must store about a CD is (see the specific data section): the CD title, the recording year, the artist that has recorded the CD, the list of songs, some comments about the CD, the front cover, the price, the number of times that the CD has been bought, the profiles of the customers that usually purchase it, the number of units in stock, and the music categories to which the CD belongs. Notice that all these features present a simple nature except from the artist of the CD

4. A Task-based Requirements Model for Specifying Web Applications

and the music categories. The nature of these features is described by the information templates whose identifier is Id02 and Id03 respectively. The information template with identifier Id02 (which is associated to the entity Artist) is shown in Figure 4.24.

Identifier:	Id1		
Entity:	CD		
Specific Data:	<i>Name</i>	<i>Description</i>	<i>Nature</i>
	Title	Title of the CD	String
	Year	Recording year of the CD	Number
	Artist	Artist that has recorded the CD	Id02
	Songs	Songs of the CD	List
	Comments	A brief commentary of the CD	Text
	Front Cover	Front cover of the CD	Image
	Price	Price of the CD	Currency
	Purchase times	Times that the CD has been purchased	Number
	Client Profiles	Profiles of the clients that have purchased the CD	List
	Stock	Quantity of units in stock	Number
	Music Categories	Category to which the CD belongs	List(Id03)

Figure 4.23 Information template associated to the entity CD

Figure 4.24 shows the information template associated to the entity Artist (identified in the description of the elementary task *Add CD*, see Figure 4.20). According to this template, the information that the system must store about an Artist is the following: The name of the artist, the nickname, the date of the artist's birth, the country where the artist was born, the record label with which the artist has signed a deal and the artist's personal Web page. All these features present a simple nature

4. A Task-based Requirements Model for Specifying Web Applications

Identifier:	Id2		
Entity:	Artist		
Specific Data:	<i>Name</i>	<i>Description</i>	<i>Nature</i>
	Name	Name of the artist	String
	Nickname	Nickname of the artist	String
	Birth Date	Date of the artist's birth	Date
	Country	Country in which the artist was born	String
	Label	Record label with which the artist has signed a deal	String
	Web Page	Web page of the artist	Url

Figure 4.24 Information template associated to the entity Artist

Several times, when we describe the specific data section of an entity, we realize that the entity is a *super-type* of other entities already described. For instance, following with the Amazon example, we have identified the entity Product in the description of the elementary task *Inspect Shopping Cart* (see Figure 4.21). When we try to describe the specific data section associated to this entity we realize that it depends on whether the product is a CD, a Book or a Software product (entities identified in the description of other elementary tasks). If the product is a CD, the specific data section associated to the entity Product is the same as the one defined for the entity CD; however, if the product is a book the specific data section associated to the entity Product is the same as the one defined for the entity Book. These situations appear when an entity is a super-type of other entities. Thus, the entity Product is a super-type of the entity CD, the entity Book and the entity Software. We indicate this aspect by referencing to the entities CD, Book and Software in the specific data section associated to the entity Product. In order to reference these entities we indicate the name of the entity and the identifier of the information template associated to the entity. We can see an example of this in Figure 4.25.

Identifier:	Id3	
Entity:	Product	
Specific Data:	Super-Type Of:	
		CD: id1
		Software: id5
		Book: id8

Figure 4.25 Information template associated to the entity Product

4. A Task-based Requirements Model for Specifying Web Applications

Description of Exchanged Data in User-System Interactions

We have seen in Section 4.2.2 an approach for describing elementary tasks. In this approach, the concept of Interaction Point (IP) has been introduced to capture the interactions between the system and the user. Each interaction between the system and the user constitutes an exchanged of data, and each IP is associated to the entity to which the data is related. In order to facilitate the construction of these descriptions, details about the exchanged data are not defined.

In this section, we propose a technique to describe the data exchanged in each IP in detail. This technique is based on the association of the features defined for each entity (in the information templates introduced above) with the different IPs defined in the descriptions of elementary tasks. To do this, we propose the use of templates for exchanged data. Figure 4.26 shows the exchanged data template associated to the IP `Output(CD,*)`, defined in the elementary task `Add CD` (see Figure 4.20). According to this template, this IP provides for each CD that is available in stock: the title of the CD, the price, and the name the artist that has recorded the CD.

Identifier:	O2	
IP:	Output(CD,*)	
Elementary Task:	Add CD	
Retrieval Condition:	self.stock > 0	
Exchanged Data:	Entity and Template Id	Feature
	CD: id1	Title
	CD: id1	Price
	CD: id1	Front Cover
	Artist: id2	Name

Figure 4.26 Exchanged Data Template associated to the IP `Output(CD,*)`

As we can see in Figure 4.26, each exchanged data template is made up of the following fields:

- *Identifier*: the identifier of the template.
- *IP* and *Elementary Task*: These fields indicate the IP in which the data is exchanged. In order to textually identify an IP, we use the following notation: *Output (Entity, Cardinality)* for Output IPs, and *Input (Entity, System Action)* for Inputs IPs. We also indicate the elementary task in whose description the IP has been defined.
- *Retrieval Condition*: This field can only be indicated for Output IPs. It allows us to define a condition that every entity provided in the Output IP must

4. A Task-based Requirements Model for Specifying Web Applications

accomplish. This allows us to filter the entities that users access in an Output IP. This type of constraints is defined by using the Object Constraint Language (OCL) [OCL]. For instance, we are indicating in Figure 4.23 that the *IP Output(CD,*)* only provides users with information about those CDs that are available in stock.

- *Exchanged Data*: This field indicates the entity features that describe the exchanged data. In the case of Output IPs, these features constitute the data that is shown to users. In the case of Input IPs, these features constitute the data that is requested to users. For each feature we indicate its name, the entity, and the identifier of the information template associated to the entity.

Features defined in this field must be related to the entity associated to the IP by following one of these criteria:

1. *The feature is directly related to the IP entity.* The feature has been defined in the information template associated to the IP entity. For instance, for the IP *Output(CD,*)* defined in the elementary task Add CD (see Figure 4.20) we can define in the exchanged data field all the features defined in the information template associate to the entity CD (see Figure 4.23).
2. *The feature is related to an entity which constitutes the complex nature of a feature that is directly related to the IP entity.* For instance, for the IP *Output(CD,*)* defined in the elementary task Add CD (see Figure 4.20) we can associate the features defined in the information template associate to the entity Artist (see Figure 4.24), because the entity CD has a feature (the artist that has recorded the CD) whose nature is the entity Artist.
3. *If the IP entity constitutes the super-type of several entities (sub-types), we define the features considering the criteria 1 and 2 for the different sub-entities.* For instance, lets consider the IP *Output(Product,1)* defined in the elementary task Inspect Shopping Cart (see Figure 4.21). This IP shows information about a product. However, this information changes depending on whether the product is a CD, a book or a software product. Thus, we must indicate (according to Rules 1 and 2) the features that are shown in the IP when the product is a CD, when the product is a Book and when the product is a software product. We can see an example of these templates in Figure 4.27.

4. A Task-based Requirements Model for Specifying Web Applications

Identifier:	O11	
IP:	Output(Product,1)	
Elementary Task:	Inspect Shopping Cart	
Retrieval Condition:	-	
Exchanged Data:		
As CD:	Entity and Template Id	Feature
	CD: id1	Title
	CD: id1	Price
	CD: id1	Year
	CD: id1	Songs
	CD: id1	Comments
	CD: id1	Front Cover
	Artist: id2	Name
	Entity and Template Id	Feature
	Software: id4	Name
	Software: id4	Price
	Software: id4	Company
	Software: id4	Image
	Software: id4	System Requirements
	Software: id4	Medium
	Software: id4	Description
	Entity and Template Id	Feature
	Book: id5	Name
	Book: id5	Price
	Book: id5	Editorial
	Book: id5	Year
	Book: id5	Summary
	Book: id5	Cover

Figure 4.27 Exchanged Data Template associated to the IP Output(Product,1)

It is possible that a super-type constitutes the nature of an entity feature. Let's consider for instance the entity Item (identified from the IP Output (Item,*) in the task Inspect Shopping Cart). An Item is created when the user adds a product to the

4. A Task-based Requirements Model for Specifying Web Applications

shopping cart. An Item is made up of the following features: the added *product*, the *quantity* of units that are added and the *total price* (that is, the quantity of units per the product price). In this case, the feature *product* has complex nature that is defined by the entity Product.

Then, we can associate features of the entity Product to the IP Output (Item,*) in the same way that we have associated features of the entity Artist to the IP Output(CD,*) (see Figure 4.26). We can do this because both entities Item and CD present a feature whose complex nature is defined by the entities Product and Artist, respectively. However, in the case of the entity Item, the complex nature of its feature is defined by an entity that is a super-type (Product is a super-type of CD, Software and Book). Thus, in order to indicate the product feature that we want show in this IP we must to distinguish among all its subtypes. Figure 4.28 shows the exchanged data template associated to the IP Output(Item,*) that illustrates this aspect. According to this template, for each Item of the shopping cart the system provides users with the quantity, the total price and: (1) the title if the associated product is a CD, (2) the name if the product is a software product or (3) the name if the product is a book.

Identifier:	O10	
IP:	Output(Item,*)	
Elementary Task:	Inspect Shopping Cart	
Retrieval Condition:	-	
Exchanged Data:	Entity and Template Id	Feature
	Item: id8	Quantity
	Item: id8	Total Price
	Product: id9	As CD: Title As Software: Name As Book: Name

Figure 4.28 Exchanged Data Template associated to the IP Output(Item,*)

4.3 A RE tool for Creating Task-based Requirements Models

In this section, we present a prototypical RE tool that supports the construction of task-based requirements models. The technology used to develop this tool is based on the Eclipse platform [Eclipse].

The basis for Eclipse is the Rich Client Platform (RCP). Eclipse employs plug-ins in order to provide all its functionality on top of (and including) the rich client platform, in contrast to some other applications where functionality is typically hard coded. The plug-in architecture supports writing any desired extension to the environment. In this context, several projects are being developed in order to extend the Eclipse

4. A Task-based Requirements Model for Specifying Web Applications

environment with plug-ins that provides support for the development of different types of software. We can find plug-ins that support the development of database systems, mobile applications, AJAX-based applications, tools for testing and so on. We can also find plug-ins that support the development of CASE modelling tools. We have used these plug-ins in order to develop the RE tool.

4.3.1 The Plug-ins used to develop the RE tool

In order to develop the RE tool we have used the following plug-ins:

- *Eclipse Modelling Framework (EMF)*: The EMF project provides us with a modeling framework and code generation facilities for building tools and other applications based on a structured data model. The core of this framework includes both a meta model (Ecore) for describing models and runtime support for managing models, including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically. Ecore is an implementation of the Essential Meta-Object Facilities (EMOF). EMOF is a subset of the standard MOF 2.0 [MOF] proposed by the Object Management Group (OMG) for describing meta-models.

Furthermore, EMF provides mechanisms to generate Java files from an Ecore metamodel. These files implement the different elements of the metamodel (by means of Java classes) providing support to create instances of them in run time (i.e. objects of a Java class that represent elements of the meta-model) as well as to manage them.

- *Graphical Editing Framework (GEF)*: The GEF project allows developers to take an existing application model and quickly create a rich graphical editor.

Basically, GEF provides an infrastructure for developing graphical editors by following the pattern model-view-controller (MVC). GEF itself provides support to develop the *controller* part. In order to develop the *model* and *view* parts GEF does not force to use specific libraries. However, the most common way of using GEF is together with Draw2D for the *view* part and EMF for the *model* part. Figure 4.29 shows how the MVC pattern is implemented with GEF.

4. A Task-based Requirements Model for Specifying Web Applications

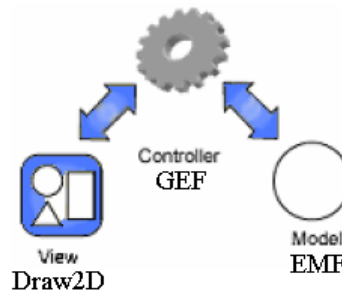


Figure 4.29 The MVC pattern with GEF

- *Eclipse Graphical Modeling Framework (GMF)*: The GMF project provides us with a generative component and a runtime infrastructure for developing graphical editors based on EMF and GEF.

GMF allows us to declaratively describe the different associations among elements of a model and their visual representation by means of models. From these models GMF automatically generates a graphical editor implemented by means of GEF. This graphical editor provides support for creating, modifying and deleting each visual representation. The use of GMF provides us with an abstract way of developing graphical editors, without the need of considering the technological aspects introduced by GEF.

In order to develop a graphical editor, GMF proposes the construction of the following models:

1. **Graphical Definition Model**: in this model we indicate the elements that must appear in the diagram as well as their graphical representation.
2. **Tool Model**: In this model, we indicate the different actions that can be called from the tool bar in order to create diagram elements. We can also define groups of actions.
3. **Association Model**: In this model, we associate elements of the Ecore meta-model with their graphical representation (defined in the Graphical Definition Model) and with their creation action of the tool bar (defined in the of the tool model).
4. **Generation Model**: The graphical editor is developed as an Eclipse plug-in. Thus, we define in this model some parameters for configuring this plug-in. These parameters are related to aspects such as name and id of the plug-in, name of its developer, the extensions of models and diagrams, whether diagrams can be printed or not, and so on.

Considering the three Eclipse plug-ins introduced above, the graphical editor for the task-based requirements model has been developed by following the next steps (see Figure 4.30):

4. A Task-based Requirements Model for Specifying Web Applications

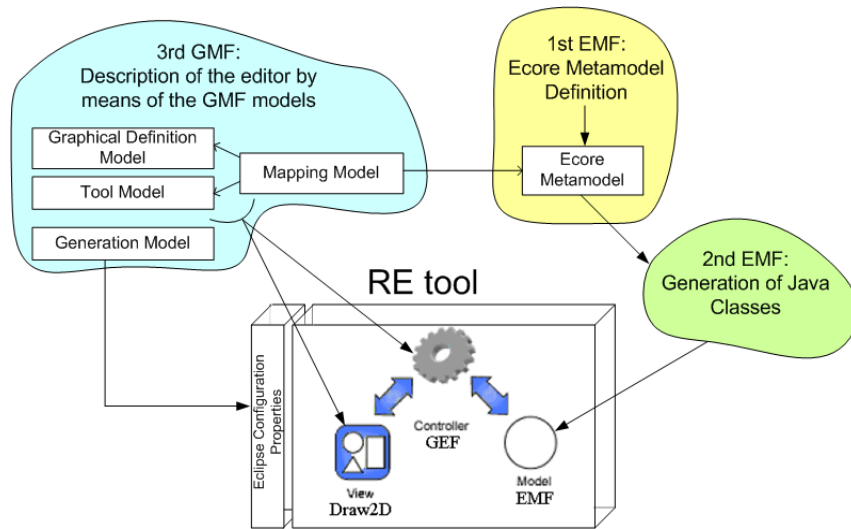


Figure 4.30 Steps for developing the graphical editor

1. First, we have described the meta-model of the task-based requirements model by means of Ecore (that is included in the EMF framework).
2. Next, we have used the facilities provided by EMF in order to automatically generate from the Ecore meta-model a set of Java classes that provides support to manage instances of the meta-model elements in run time.
3. The Java classes generated by EMF constitute the *model* part of model-view-controller architecture in which the graphic editor is implemented. In order to implement the *view* and *controller* parts we have used GMF. By defining the models proposed by this plug-in we have automatically generated: (1) GEF-based code that implements the controller part and (2) Draw2D code that implements the *view* part.

Model Persistence

One of the most important benefits of using EMF is the great support that this plug-in provides for storing models in a persistence way. The framework provides valuable mechanisms based on XML Metadata Interchange (XMI) in order to manage the persistence of objects. In this context, XMI is the standard used to store models. The structure of a XMI document is very similar to the structure of the model that it represents. The XMI document uses the same names and the same element hierarchy as the model. This aspect makes the relation between a model and its serialization easy to understand.

4. A Task-based Requirements Model for Specifying Web Applications

Figure 4.31 shows a partial view of a XMI document in which a task-based requirements model is stored. In particular, we can see a partial view of the serialization of the task taxonomy presented in Section 4.2.2 (see Figure 4.5).

```
<Task xsi:type="Non-ElementaryTask" name="Purchase Products">
  <StructuralRefinement StructuralRefinementFrom="//@Task.0"
    StructuralRefinementTo="//@Task.1"/>
  <StructuralRefinement StructuralRefinementFrom="//@Task.0"
    StructuralRefinementTo="//@Task.9"/>
</Task>
<Task xsi:type="Non-ElementaryTask" name="Buy Products">
  <TemporalRefinement TemporalRefinementFrom="//@Task.1"
    TemporalRefinementTo="//@Task.2"/>
  <TemporalRefinement TemporalRefinementFrom="//@Task.1"
    TemporalRefinementTo="//@Task.3"/>
</Task>
<Task xsi:type="Non-ElementaryTask" name="Collect Products">
  <TemporalRelationship TemporalRelationshipTo="//@Task.3"
    TemporalRelationshipFrom="//@Task.2"/>
  <TemporalRefinement TemporalRefinementFrom="//@Task.2"
    TemporalRefinementTo="//@Task.4"/>
  <TemporalRefinement TemporalRefinementFrom="//@Task.2"
    TemporalRefinementTo="//@Task.8"/>
</Task>
<Task xsi:type="ElementaryTask" name="Checkout"/>
<Task xsi:type="Non-ElementaryTask" name="Add Products to Shopping Cart">
  <TemporalRelationship ConcurTaskTreeTemporalRelationship="|">
    TemporalRelationshipTo="//@Task.8"
    TemporalRelationshipFrom="//@Task.4"/>
  <StructuralRefinement StructuralRefinementFrom="//@Task.4"
    StructuralRefinementTo="//@Task.6"/>
  <StructuralRefinement StructuralRefinementFrom="//@Task.4"
    StructuralRefinementTo="//@Task.7"/>
  <StructuralRefinement StructuralRefinementFrom="//@Task.4"
    StructuralRefinementTo="//@Task.5"/>
</Task>
<Task xsi:type="ElementaryTask" name="Add a CD"
  Frequency="100 times per hour">
  <Users>Visitor</Users>
  <Users>Customer</Users>
</Task>
...
```

Figure 4.31 Partial view of the task taxonomy XMI serialization

4.3.2 The RE tool user interface

In this section, we explain the main characteristics of the RE tool user interface, explaining how the different parts of the task-based requirements model are created.

As explained above, the RE tool has been developed as an Eclipse plug-in. In this context, the main characteristics of its user interface are the same as the characteristics of the Eclipse environment. Figure 4.32 shows a snapshot of the

4. A Task-based Requirements Model for Specifying Web Applications

graphical editor incorporated in the Eclipse environment. Its user interface is divided in four main frames: (1) Frame 1 is the file explorer, which allows us to directly access the different files that constitute a requirements specification project; Frame 2 is the modelling zone, where we can graphically create the different parts of a task-based requirements model; Frame 3 is the tool bar, which provides us with tools for creating the different graphical elements of a diagram; finally, Frame 4 is the property editor, which allows us to specify the textual properties associated to the graphical elements.

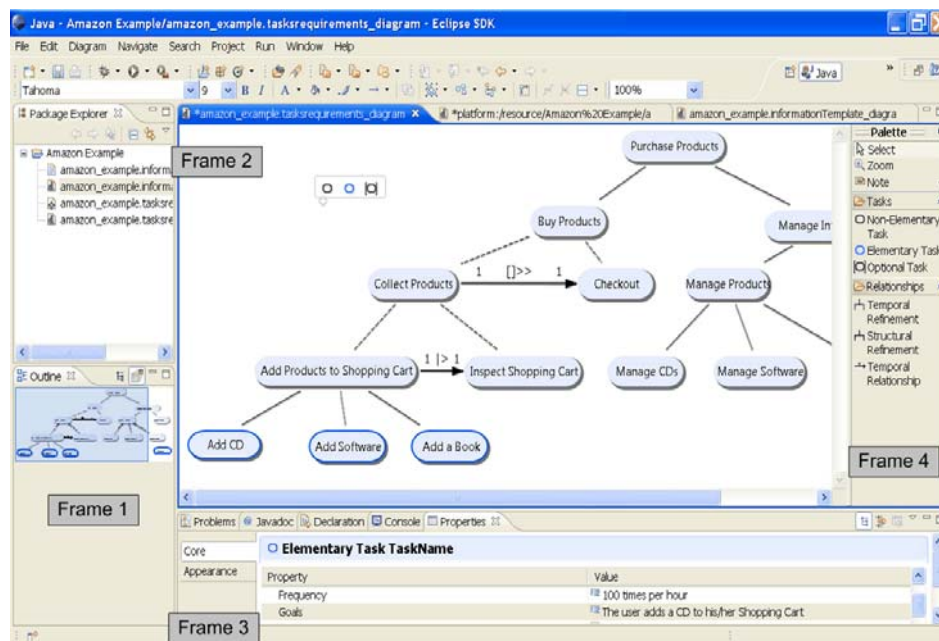


Figure 4.32 Eclipse-based user interface of the graphical editor

The modelling zone included in the RE tool interface allows us to create the different parts of a task-based requirements model. However, when a requirements specification project is newly created the RE tool only provides support to create the task taxonomy. Support for creating task performance descriptions and information templates is available when elementary tasks are created in the task taxonomy. Figure 4.33 shows the modelling zone with a task taxonomy (the task taxonomy of the Amazon example) together with the tool bar that allows us to create it. Notice how the tool bar provides us with every visual metaphor that is required to create the task taxonomy: elementary tasks and non-elementary tasks; structural refinements and temporal refinements; and finally, temporal relationships between tasks. Figure 4.33 also shows the property editor that appears when we click over an elementary task (in this case, the elementary task Add CD). This editor allows us to define the different properties that characterize an elementary task: name, goal, users, frequency and additional constraints (they appear in alphabetical order).

4. A Task-based Requirements Model for Specifying Web Applications

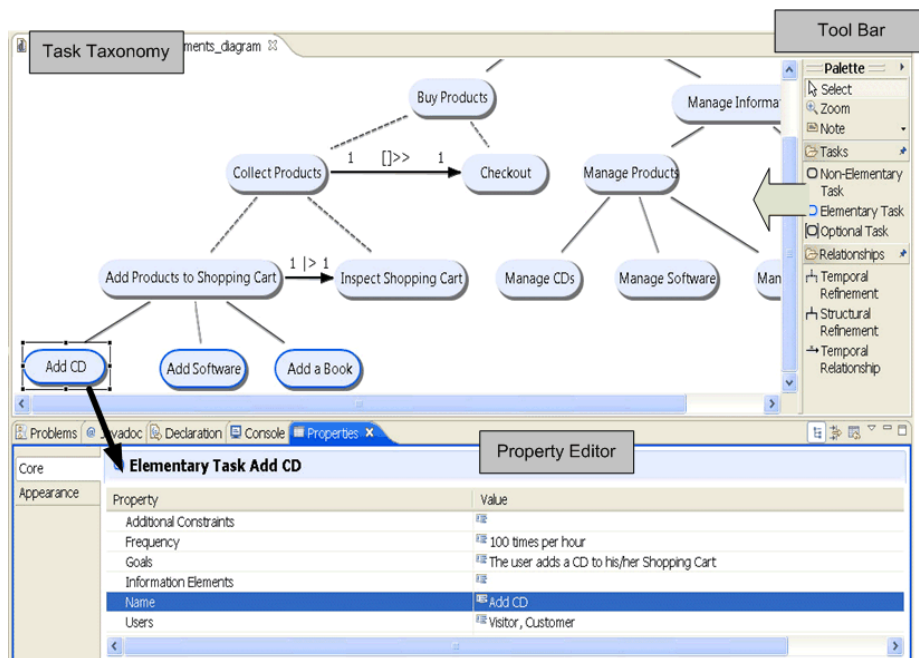


Figure 4.33 Modelling zone, tool bar and property editor for creating task taxonomies

Once elementary tasks are defined we can create the activity diagrams that represent their performance. To do this, we just need to double click over an elementary task. After this, the RE tool provides us with a new diagram in the modelling zone (in which we can create the task performance description associated to the elementary task that we have selected), with a new tool bar (which provides us with the required visual metaphors for creating a task performance description), and with a new property editor (which allows us to define the textual properties of the elements that define a task performance description). Figure 4.34 shows the modelling zone in which the activity diagram associated to the task Add CD is created. This figure also shows the tool bar that allows us to create elements such as Output IPs, Input IPs or System Actions as well as different arcs to connect these elements (flow of steps with and without conditions). Figure 4.34 also shows the property editor that appears when we click over an element of the task performance description. In this case, we show the property editor that appears when we click over an Output IP. This editor allows us to define the properties of an Output IP: cardinality and entity.

4. A Task-based Requirements Model for Specifying Web Applications

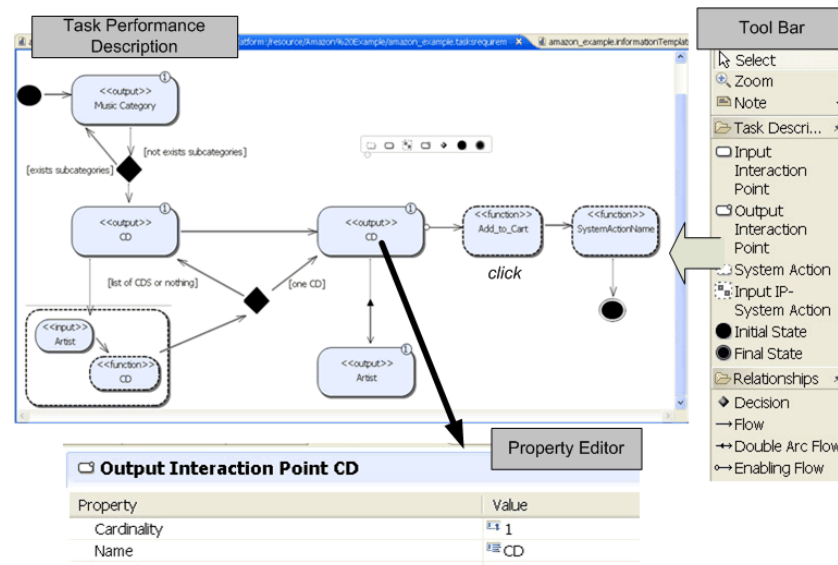


Figure 4.34 Modelling zone, tool bar and property editor for creating task performance descriptions

We have seen how to use the RE tool in order to create a task taxonomy and its associated task performance descriptions. However, in order to create a complete task-based requirements model we also need to create information templates (which characterize the information that the system must store about the different entities) and data exchanged templates (which indicate the data that is shown or requested by the system in each interaction point). The graphical editor also provides us with support for creating these templates.

In order to create the information templates associated to the different entities the RE tool allows us to open a new diagram in the modelling zone. In this diagram we can graphically specify the information templates that must be created as well as the features of each template. Figure 4.35 shows an example where the features of a CD and Artist are defined.

Finally, once we have created the information templates we can associated their features to the different interaction points defined in the task performance descriptions. To do this, we must create data exchange templates. To create these templates we must access a diagram where a task performance description is defined. Then, we just need to click over the IP to which the data exchange template must be associated. When we do this, the corresponding property editor appears (presented in Figure 4.34). This editor has two additional properties (that have not been presented above) that allow us to indicate the entity features that must be shown/requested in the IP as well as a retrieval condition (in the case of Output IPs). These properties are shown in Figure 4.36.

4. A Task-based Requirements Model for Specifying Web Applications

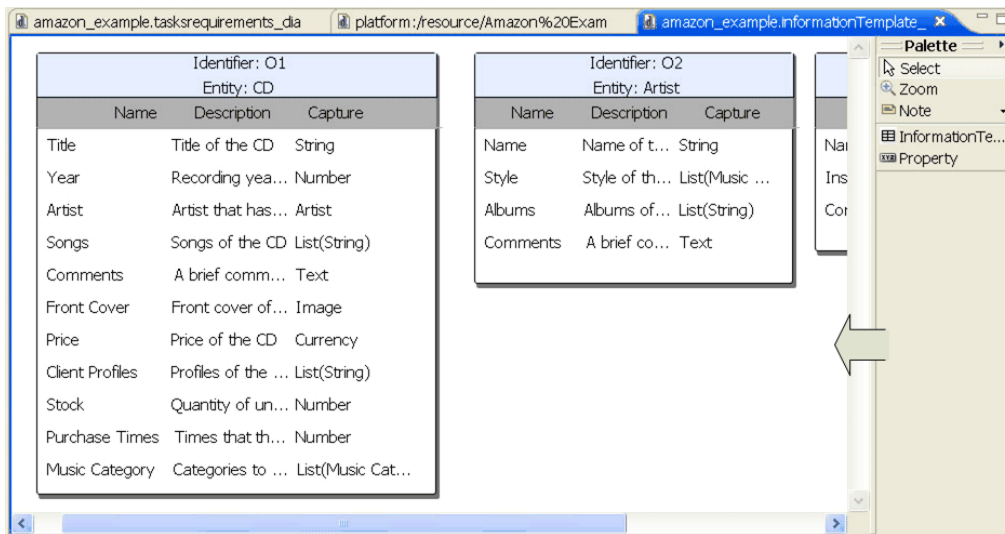


Figure 4.35 Modelling zone and tool bar and for creating information templates

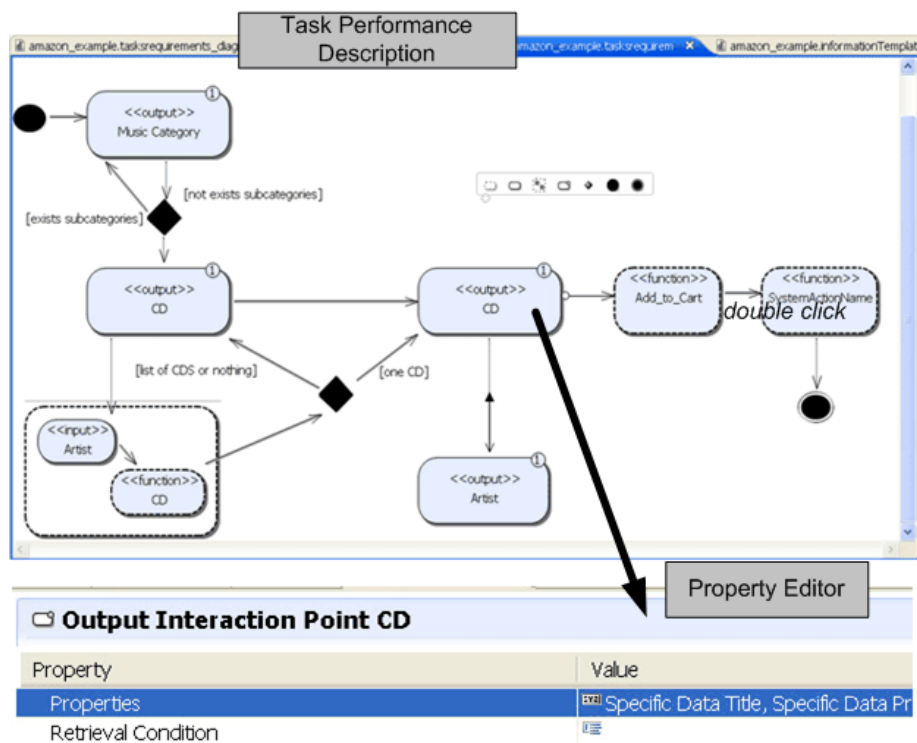


Figure 4.36 Modelling zone and property editor for creating exchanged data templates

4.4 Conclusions

In this chapter, we have introduced a requirements model based on the concept of task for the specification of Web applications requirements. In order to construct this model a top-down strategy has been proposed:

1. We start providing a general description of the Web application through the statement of purpose.
- 2.1 From this general description a Hierarchical Task Analysis is performed in order to identify the tasks that represent the user's needs. To do this, we have provided guidelines in order to both identify the most general task and refine this task into more specific ones (defining a task taxonomy).
- 2.2 Next, we have introduced a technique in order to describe elementary tasks. It constitutes a novel technique where system and user actions are combined with aspects related to the interaction between both. To do this, we have introduced the concept of interaction point.
- 3 Finally, we have presented mechanisms based on templates in order to detail the information that the system must store as well as the information that user and the system exchange in the performance of the different tasks.

By constructing this task-based requirements model, three types of requirements (see Chapter 2) can be captured:

- Transactional requirements, which express what the Web application has to compute internally. They are captured by means of the system actions that are defined in the performance descriptions of elementary tasks.
- Data requirements, which establish the information that is stored and administrated by the Web application. They are captured by means of the information templates that are associated to the entities identified in the performance descriptions of elementary tasks.
- Navigational requirements, which represent users' navigation needs through the hyperspace. They are captured by means of temporal relationships between tasks in the task taxonomy as well as the interaction points defined in performance descriptions of elementary tasks. Furthermore, details about the information that users navigate are captured in the exchanged data templates.

The meta-model of the task-based requirements model can be found in Appendix A.

4. A Task-based Requirements Model for Specifying Web Applications

Furthermore, we have also presented a RE tool that provides support for creating task-based requirements models. By using the Eclipse platform we have developed a tool that allows us to graphically create the different parts of a task-based requirements model: task taxonomies, task performance descriptions and both information and exchanged data templates. To do this, different diagrams are provided together with different tool bars and property editors.

The task-based requirements models created by this tool are stored in XMI. The use of this XML-based technique for storing models facilitates the interoperability of the RE tool with the tools that support the model transformation strategy presented in Chapter 6.

Chapter 5

*“One must always maintain one’s connection to the past
and yet ceaselessly put away from it”*

Gaston Bachelard
(French Philosopher and Poet, 1884-1962)

5 Requirements Traceability

In this chapter, we present a set of traceability rules that specify structured mechanisms to analyze task-based requirements models in order to generate the main components of a Web application conceptual model. In order to define the Web application conceptual model the Web engineering method OOWS [Fons et al 2003] has been used. An overview of this method can be found in Appendix B.

Although it is not possible to generate complete OOWS conceptual models from the task-based requirements model, the resultant conceptual model *skeleton* represents the main framework that the software engineer should refine and complete to have a precise representation of the Web application that is going to be developed. In particular, the traceability rules presented in this chapter allow us to partially derive (1) the structural model of the OOWS method which represents the static structure of a Web application and (2) the navigational model of the OOWS method which represents the navigational structure of a Web application.

Traceability rules are presented in an intuitive way in order to facilitate its understanding. They have been however formalized by means of graph transformations. This formalization can be found in [Valderas 2007b]. Additionally, these graph transformations allow us to define a strategy for applying traceability rules automatically. This strategy is presented in Chapter 6.

5. Requirements Traceability

Finally, we want to remark that we have chosen the OOWS method to define Web application conceptual models for two main reasons:

- This Web Engineering method has been developed in the same research group to which belongs the author of this thesis. In fact, the author has participated actively in the conception of part of the method. Thus, the author has a great knowledge about this method. This aspect has facilitated him to perform an exhaustive analysis of the best way of supporting the requirements model abstractions with the OOWS conceptual primitives.
- The OOWS code generation strategy, which allows us to define a process for automatically generating Web application prototypes from requirements models (we have explained it in Chapter 3).

This chapter is organized as follows: Section 6.1 introduces the traceability concept and the different type of traceability proposed throughout the published literature. Section 6.2 introduces some rationale about how the traceability rules are defined. Section 6.3 presents the traceability rules catalogue. They are applied to the requirements model presented in the previous chapter in order to show some examples of how the OOWS conceptual model is obtained. Finally, Section 6.4 concludes this chapter.

5.1 Traceability in Requirements Engineering

Requirements traceability can be defined as the ability to describe and follow how requirements are translated to other artefacts along the development life cycle of a software system. Requirements traceability is widely considered to be an important factor for achieving both an efficient software project management and software systems quality. Traceability allows us to:

- Validate whether or not requirements are all supported, and whether the implementation compliant with the requirements.
- Understand the user need that is addressed by each requirement.
- Verify the necessity of each requirement and how it is implemented. We also can check how design decisions affect the implementation of each requirement.
- Check how each requirement has been interpreted by programmers.
- Establish the impact of changing a requirement on software artefacts.

5. Requirements Traceability

Next, we present the definition of traceability that is proposed by the IEEE Standard-830-1998 [IEEE 1998]:

"A software requirements specification is traceable if (i) the origin of each of its requirements is clear and if (ii) it facilitates the referencing of each requirement in future development or enhancement documentation".

Furthermore, the standard indicates that good traceability practices allow for bidirectional traceability meaning that the traceability chains can be traced in both the forwards and backwards directions as shown in Figure 5.1.

- Forward traceability looks at both tracing the requirements source to the resulting requirements and tracing the resulting requirements to the work products that implement them.
- Backward traceability looks at both tracing each work product back to its associated requirements and tracing each requirement back to its source.

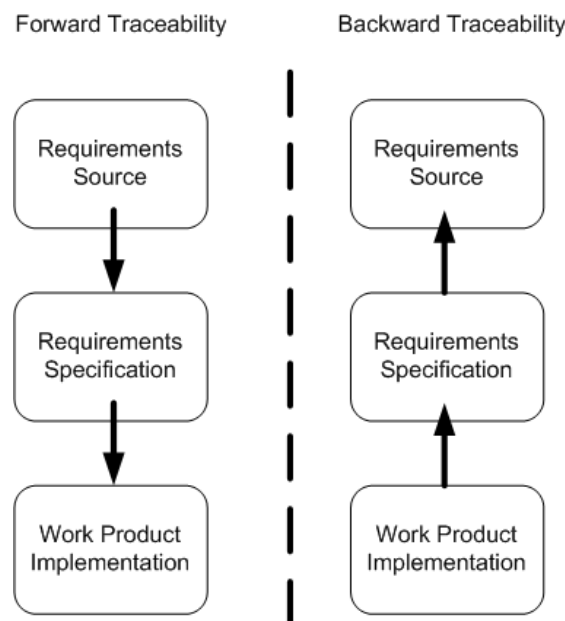


Figure 5.1 Bidirectional Traceability

Based on this bidirectional aspect, Gotel defines requirements traceability [Gotel 1995] [Fowler 1997] in the following way:

"Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forward and backward direction (i.e., from its origins,

5. Requirements Traceability

through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases)".

In this context, traceability is classified according to the direction in which the life of a requirement is followed (forward and backward). However, this is not the only criterion that can be used to classify traceability. Next, we present other classifications of traceability.

5.1.1 Other Traceability Classifications

Two different types of traceability are distinguished according to the classification proposed by Gotel [Gotel 1995]:

- *Pre-requirements specification (pre-RS) traceability* is concerned with those aspects of a requirement's life prior to its inclusion in the RS (requirement production). This type of traceability is used to track the relationship between each requirement and its source. For example, a requirement might trace from a business need, a user request, a business rule, an external interface specification, an industry standard or regulation, or to some other source.
- *Post-requirements specification (post-RS) traceability* is concerned with those aspects of a requirement's life that result from its inclusion in the RS (requirement deployment). This type of traceability is used to track the relationship between each requirement and the work products to which that requirement is allocated. For example, a requirement might trace to one or more architectural elements, detail design elements, object/classes, code units, tests, user documentation topics, and/or even to people or manual processes that implements that requirement.

In this thesis, we focus on *Post-RS traceability*. We introduce model-to-model transformation rules to derive in a traceable way OOWS conceptual models from the task-based requirements model presented in the previous chapter. In this context, we know that two or more conceptual models can give support to a same requirements specification [Fowler 1997]. Thus, if we consider that we derive conceptual models from requirements, one requirement might be derived into more than one conceptual element, and all of them correctly support the requirement. However, might be other requirements that only can be derived into a unique conceptual element in order to be correctly supported. In order to consider this type of relationship between requirements and conceptual elements another traceability classification is proposed in [Einsfrán 2003]:

- *Weak traceability*, which is concerned with those aspects of the requirements model that will be translated to elements in the conceptual model but that can

5. Requirements Traceability

be changed or even deleted on destination. This means that aspects with weak traceability will only be *proposed elements* in the conceptual model.

- *Strong traceability*, which is concerned with those aspects of the requirements model that will be translated to elements into the conceptual model that cannot be changed or deleted on destination. This means that aspects with strong traceability will be *mandatory elements* in the conceptual model.

5.2 Defining Traceability Rules: A General View

In this chapter, we present traceability rules that analyze task based requirements model to obtain OOWS conceptual model. In particular, the OOWS conceptual models that are obtained are two: the structural model and the navigational model.

On the one hand, the **OOWS structural model** (see Appendix B) captures the static structure of a Web application from a set of classes (with its operations and attributes) and the relationships between these classes. Classes represent at the conceptual model objects of the real word that are related with the Web application domain. These objects constitute the artifacts that are managed by users during their activities in an organization. At the requirements level, these objects are represented by the different entities that the system must recognize in order to properly support the tasks that users must perform. Thus, these entities constitute a valuable source of information in order to derive the classes that must be defined in the OOWS structural model.

Class attributes are properties of the real objects that the classes represent. These properties are identified at the requirements level when we specify the data requirements, that is, the detailed information that the Web application must store about each entity (information templates). Analyzing the information templates we can derive class properties. As far as relationships between classes, they can be derived by analyzing how the system store properties of entities (e.g. the fact that the nature of an entity feature depends on other entity constitutes a clear indicator that a relationship may be defined) or by analyzing how the different entities are handled by users during the performance of a task (e.g. if users needs to manage two different entities in order to perform a specific task, then these entities may require to be related). Finally, the analysis of how entities are managed during a task can be also used to derive operations. For instance, the fact that the user or the system performs some actions with a specific entity (for instance, adding a product to the shopping cart or updating the stock of products) may indicate the necessity of create operations that support these actions.

On the other hand, the **OOWS navigational model** (see Appendix B) captures the navigational structure of a Web application. This structure indicates how the information and functionality of the Web application is organized in nodes and links

5. Requirements Traceability

in order to be accessed by users. In this context, the way in which users interact with the Web application to perform each task constitutes a valuable source of information for deriving the navigational structure. By analyzing each time that users interact with the Web application (interaction points) we can guess the nodes in which information must be organized to support this interaction. In the same way, by analyzing the path that users follow to perform these interactions we can derive the way in which nodes must be connected. Furthermore, the different temporal relationships defined between tasks also provide us with information to create the proper navigational structure. For instance, if a task can only be performed after another task, nodes and links must be defined in order to properly support this constraint. Finally, the OOWS navigational model is not only defined from a set of nodes and links but also from the information that is included in each node. In this context, the information that users and the system exchange in each interaction (described in detail by exchanged information templates) may be used to derive it.

The set of traceability rules presented next explain all these aspects in detail. These rules constitute a particular interpretation of task based requirements models in order to obtain OOWS conceptual models. Although some derived conceptual primitives are mandatory (strong traceability rules) in order to properly support requirements at the conceptual model, other derivations are marked as just proposed (weak traceability rules) indicating that analysts can change them if they consider it to be proper.

5.3 Traceability Rules Catalogue

In this section, we introduce the set of rules that summarizes the traceability structures that we create when moving from a task-based requirements model to an OOWS conceptual Model. This catalogue is under continuous study and revision in accordance with the experience we acquire when applying it to different Web development projects.

Each traceability rule is defined by means of the following structure:

- *Task Description Element (TDE)* – element(s) defined in a task-based requirements model.
- *Maps to* – primitive(s) of the OOWS conceptual model that corresponds to the element(s) of the task-based requirements model identified above.
- *Except When* – situations (if there are any) in which this rule must not be applied.
- *Traceability Type* – indicates if the traceability of the rule is *strong* or *weak*.

5. Requirements Traceability

These rules are grouped in two main sets:

1. Rules that are applied to the requirements model in order to derive the OOWS structural model. These rules derive conceptual primitives such as classes, relationships between classes or transactions. In order to easily identify and reference these rules, they are numbered by means of the Si notation, where S indicates that it is a rule for the structural model definition and the i indicates a rule number.
2. Rules that are applied into the requirements model in order to derive the OOWS navigational model. These rules derive conceptual primitives such as user types, navigational contexts, navigational links or index and filters. These rules are numbered by means of the Ni notation, where N indicates that it is a rule for the navigational model definition and the i indicates a rule number.

Next, we study each of these two groups of rules in detail.

5.3.1 Structural Model Traceability Rules

In this section, we present a set of traceability rules that allows us to systematically derive the OOWS structural model of a Web application (see Appendix B). These rules are classified into three types:

1. Class Rules: These rules allow us to derive the different classes that define the structural model as well as the attributes and operation of these classes.
2. Relationship Rules: These rules allow us to derive the different relationships that must be defined among the identified classes.
3. Transaction Rules: These rules allow us to derive local transactions. These transactions must be included in the definition of classes.

In order to explain each rule, we present first a brief rationale about the rule definition. Next, the rule itself is presented. Finally, an example of the rule application is shown.

5. Requirements Traceability

1. Class Rules

In this group of rules four traceability rules are included. We explain each of them next.

Rule S1: Entity to Classes

Rationale: Entities represent objects of the real word that participate in the performance of a task. At the requirements model, entities also indicate the information that the system must manage. Entities are associated to either IPs or search system actions. On the one hand, IPs represent steps where the system exchanges information with the user, o vice versa. This information is related to an entity. On the other hand, search system actions represent steps in which the system inquiries specific information. This information is also related to an entity. Then, the information that the system handles is always represented by entities at the requirements level. At the conceptual level, the information that the system must handle is specified in the structural model by means of classes. Thus, for each entity defined in a task performance description a class in the structural model is derived. Rule S1 capture this derivation.

We have defined Rule S1 with a strong traceability. Then, in order to correctly perform each task described in the requirements model, the system has to store information about each class derived by Rule S1 by imperative.

The Rule:

Rule S1:

TDE: Each *entity* associated to an IP or a search system action

Maps to: a class

Traceability: Strong

Some Examples: Figure 5.2 graphically shows the classes that are obtained when Rule S1 is applied to the task Add CD. These classes are: Music Category, CD and Artist. According to this figure, all classes are derived from Output IPs. However, the classes Artist and CD can be also derived from the Input IP or the search system action respectively.

Figure 5.3 shows another example of the application of Rule S1. In this case, the classes Item and Product are derived from the performance description of the task Inspect Shopping Cart.

5. Requirements Traceability

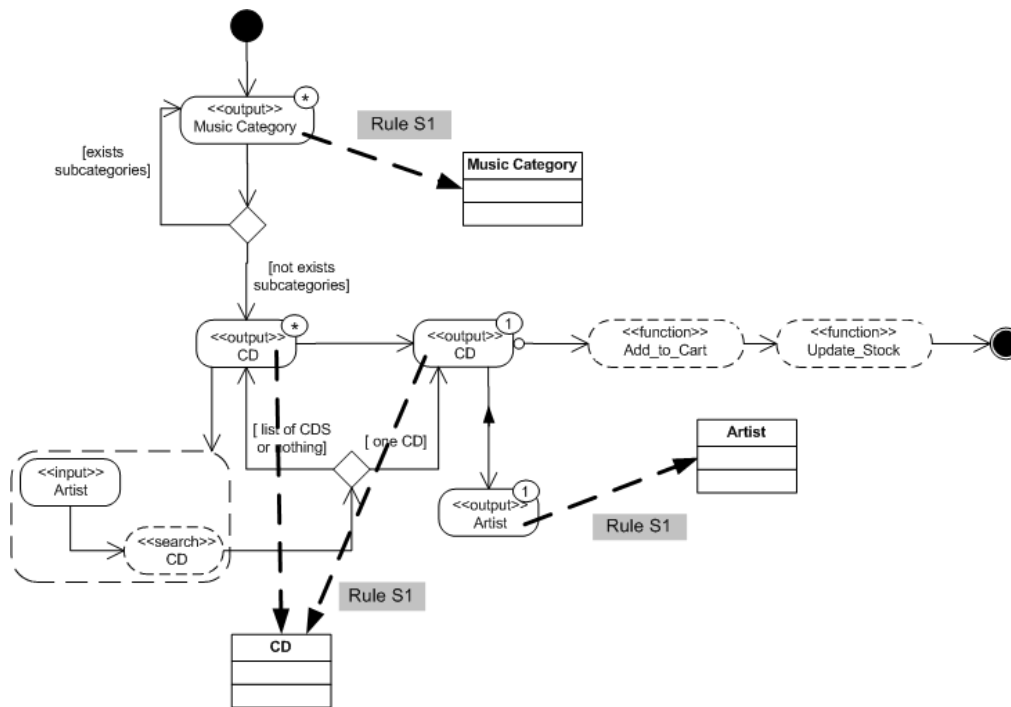


Figure 5.2 Example of application of Rule S1

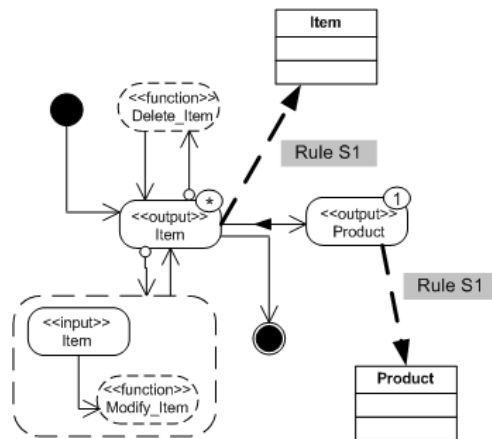


Figure 5.3 Another example of application of Rule S1

5. Requirements Traceability

Rule S2: Entity Features to Class Attributes

Rationale: Once classes have been derived, we must obtain the attributes and operations that characterize them. Classes in the structural model represent objects of the real world that are related to the Web application domain. Class attributes describe characteristics of these objects. At the requirements level, real world objects are represented by entities, and the characteristics of these objects are described by the features associated to the entities (by means of information templates). Then, in order to derive class attributes we have defined Rule S2. This rule derives a class attribute for each feature associated to the entity from which the class is derived (by means of Rule S1). Rule S2 is only applied to the entity features that have a simple nature. In order to handle complex nature features Rule S6 has been defined (further explained).

We have defined Rule S2 with a weak traceability. This means that class attributes are proposed elements and they can be deleted, modified or added new ones without altering the task logic in a critical way.

The Rule:

Rule S2:

TDE: Each *feature* that is associated to an entity by means an information template, and its nature is simple,

Maps to: an attribute of the class derived from the entity

Traceability: Weak

Some Examples: Figure 5.4 shows the application of Rule S2 to the information template associated to the entity CD. We can see how each feature that has a simple nature (all except from the feature Artist) is derived into an attribute of the class CD.

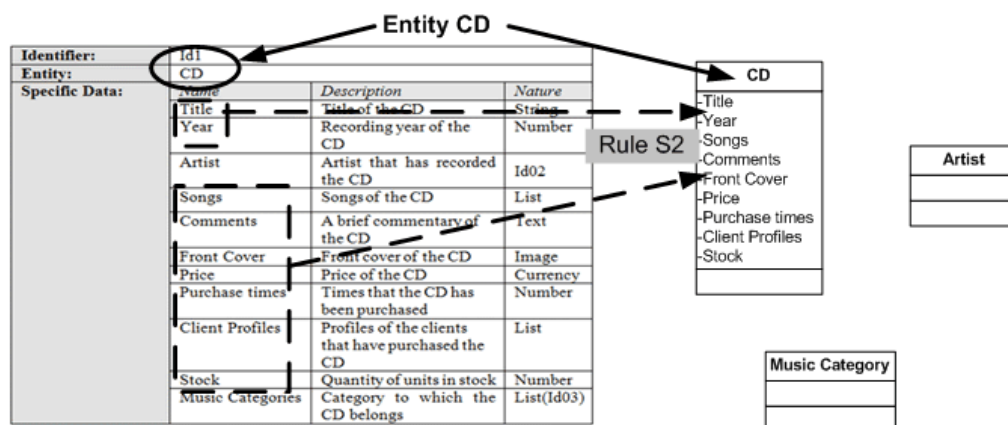


Figure 5.4 Example of application of Rule S2

5. Requirements Traceability

Figure 5.5 shows how the attributes of the class Item are derived by applying the Rule S2.

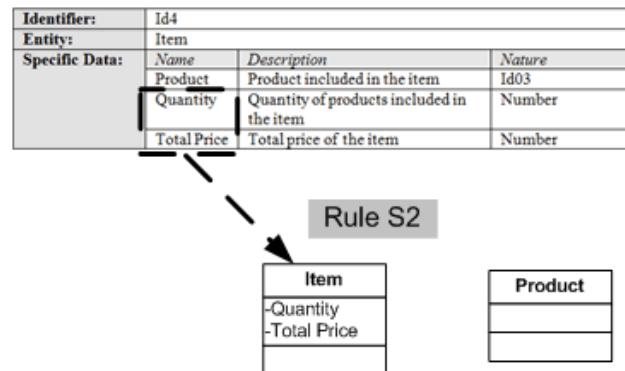


Figure 5.5 Another example of application of Rule S2

Rule S3a: Function System Actions to Class Operations

Rationale: Class operations are mechanisms that allow the system to manage objects that belong to classes. At the requirements level, we describe mechanisms that allow the system to manage objects (those represented by entities) by means of function system actions. In particular, these mechanisms are defined by those function system actions that are connected to an Output IP and moreover they are related to the IP entity (the arc's source is depicted with a small circle, see Section 4.2.2). Rule S3a derives a class operation for each of these function system actions. The class in which the operation is defined is the class derived from the IP entity (by means of Rule S1).

We have defined Rule S3a with a strong traceability. Thus, in order to correctly perform each task described in the requirements model, the system has to allow users to activate the derived operations as imperative.

The Rule:

Rule S3a:

TDE: Each *function system action* that: (1) is connected to an Output IP and (2) is related to the IP entity

Maps to: an operation that is defined in the class that has been derived from the Output IP entity

Traceability: Strong

An Example: Figure 5.6 shows the application of Rule S3a to the performance description of the task Add CD. In this case, an operation in the class CD is derived from the function system action Add_to_Cart. The derived operation belongs to the

5. Requirements Traceability

class CD because this is the class that has been derived from the entity associated to the IP Output (CD,1) (IP from which the function system action is activated). Finally, notice how the function system action is related to the IP entity (the arc's source is depicted with a small circle).

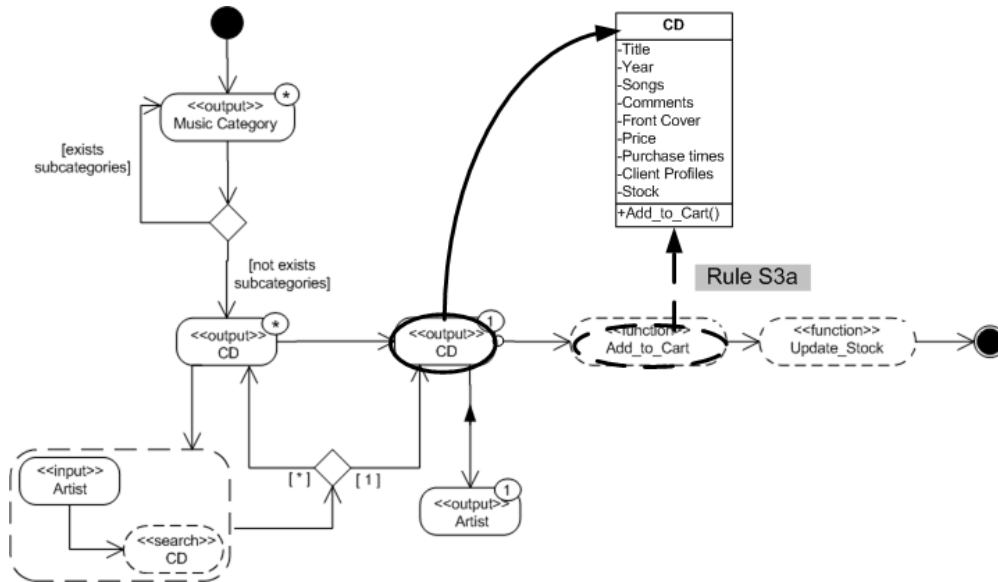


Figure 5.6 Example of application of Rule S3a

Rule S3b: Exchanged Data to Class Operation Parameters

Rationale: The example in Figure 5.6 does not show how class operation parameters are derived because the operation *add_to_cart()* does not need parameters. However, other operations do. At the conceptual level, parameters define data that the system needs to perform an operation. At the requirements level, when the system requires data to perform a system action, an Input IP is defined. Class operations are derived from function system actions (see Rule S3a). Then, a class operation will require parameters if the function system action from which it is derived is connected to an Input IP. In this context, in order to identify the parameters of a class operation we need to analyze the entity features that are requested in the Input IP connected to corresponding function system action (by analyzing the corresponding data exchanged template, see Chapter 4.2.3). This aspect is captured by Rule S3b.

We have defined Rule S3b with a weak traceability. This means that operation parameters are proposed elements and they can be deleted, modified or added new ones without altering the task logic in a critical way.

5. Requirements Traceability

The Rule:

Rule S3b:

TDE: Each feature defined in a *data exchanged template* that is associated to an Input IP, and this Input IP: (1) is connected to a function system action and (2) this action has been derived into a class operation,

Maps to: a parameter of the class operation.

Traceability: Weak

An Example: Figure 5.7 shows an example of the application of Rule S3a and Rule S3b. Both rules are applied to the performance description of the task Inspect Shopping Cart. In this case, the operations *modify_item()* and *add_item()* are defined in the class *Item* by means of Rule S3a. Furthermore, the parameters of the method *modify_item()* are identified from the features associated to the Input IP that is connected to the function system action *Modify_Item*.

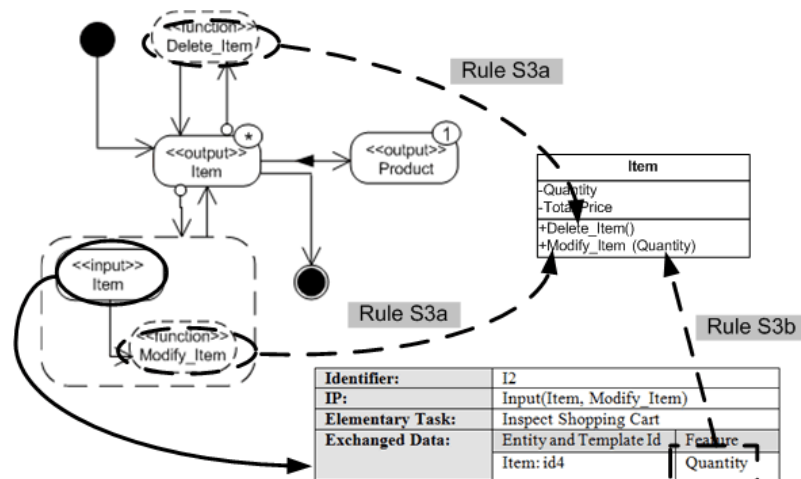


Figure 5.7 Example of application of Rule S3a and Rule S3b

2. Relationship Rules

Once classes and their definition (attributes and operations) have been derived, the next step consists in deriving relationships between classes. To do this, we must analyze information templates as well as performance task descriptions. Two types of relationships are derived:

- Association relationships. In order to derive relationships of this type three rules have been defined: Rule S4, Rule S5 and Rule S6.
- Generalization relationships. This type of relationship is derived by Rule S7.

5. Requirements Traceability

Rule S4: Output IP connections to Association Relationships

Rationale: At the conceptual level, association relationships describe some type of relation among world real objects. Some of these relations can be identified by analyzing the objects that are handled in the different activities performed in an organization. If two types of objects are used in the same activity, their corresponding classes may need to be related. For instance, we can decide to relate an invoice with a delivery note because when people create invoices they need to collect delivery notes.

In a similar way, we can derive possible association relationships from the way in which users access information when performing a task. If users access information related to two different entities during the same task, the classes derived from these entities may be need to be related. In particular, we create an association relationship when users access information of an entity and this information provides users with the possibility of access information of other entities. In terms of task performance descriptions, we derive association relationships from arcs that connect two Output IPs that are associated to two different entities. These arcs represent the selection of information by the user: by selecting information related to the entity of one IP, users access another IP where information related to a different entity is provided. Then, we consider that the classes derived from these two entities need to be related in the structural model. This derivation is captured by Rule S4.

Rule S4 has a weak traceability. This means that the derived relationships are proposed elements. Software engineers are free to delete, modify or create new relationships. For instance, software engineers can decide to change an association relationship with another relationship with a stronger semantic such as an aggregation or composition relationship (which cannot be systematically derived from task performance descriptions, further explained).

The Rule:

Rule S4:

TDE: Each arc in a task performance description that *connect two Output IPs* that are associated to two different entities

Maps to: an association relationship between the classes derived from the two entities.

Traceability: Weak

An Example: Figure 5.8 shows the application of Rule S4 to the task Add CD. An association relationship is defined between the classes Music Category and CD. This relationship is derived from the connection between the IP Output(Music Category,*) and the IP Output (CD,*).

5. Requirements Traceability

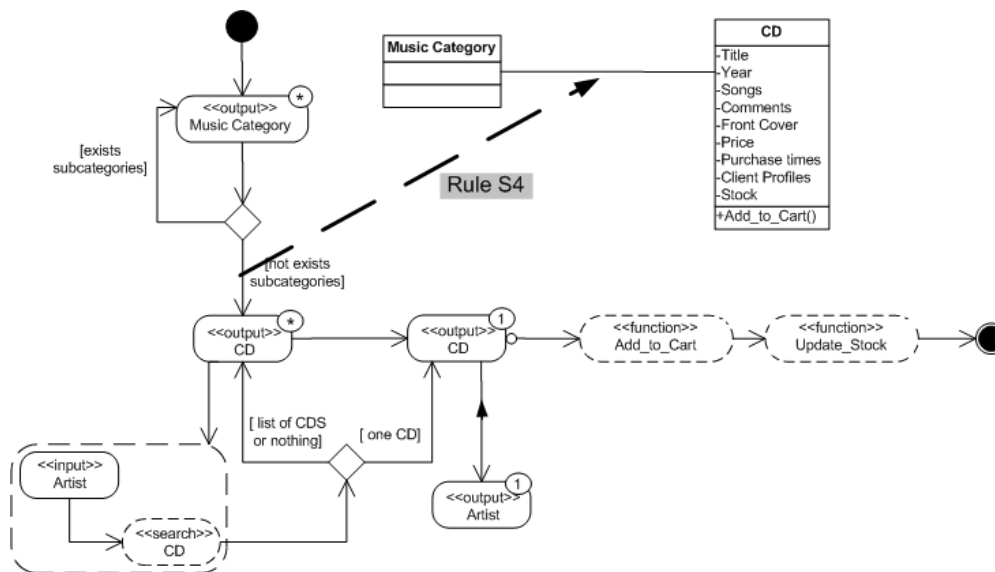


Figure 5.8 Example of application of Rule S4

Rule S5: Input IP-Search System Action connections to Association Relationships

Rationale: Association relationships can also be identified from the information that users exchange with the system in order to allow the system to perform search actions. In particular, we derive association relationships from the situations in which the system requires information about an entity (that must be introduced by users) in order to search information about a different entity. We consider that the classes derived from these two entities need to be related in the structural model. In terms of task performance descriptions, we derive an association relationship from arcs that connect an Input IP to a search system action, when these two nodes are associated to two different entities. This derivation is captured by Rule S5.

As happens with the rule presented above, Rule S5 has a weak traceability. This means that the derived relationships are proposed elements. Software engineers are free to delete, modify or create new relationships.

The Rule:

Rule S5:

TDE: Each arc in a task performance description that *connect an Input IP to a search system action*, and both nodes (IP and system action) are associated to two different entities

Maps to: an association relationship between the classes derived from the two entities.

Traceability: Weak

5. Requirements Traceability

An Example: Figure 5.9 shows an association relationship between the classes CD and Artist. This relationship is derived from the connection between the Input IP and the search system action defined in the performance description of the task Add CD. This relationship could also be derived from the connection between the IP Output(CD,1) and the IP Output(Artist,1) by means of Rule S4.

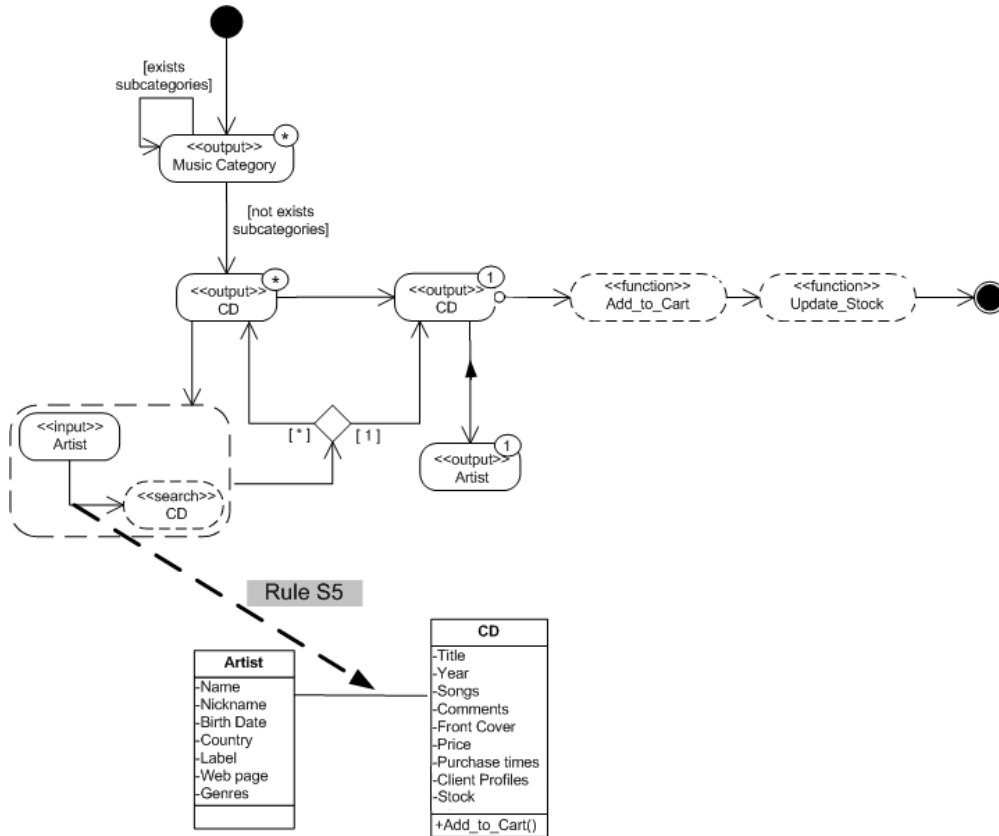


Figure 5.9 Example of application of Rule S5

Rule S6: Complex Nature Features to Association Relationships

Rationale: Another possibility that we propose to derive association relationships is to analyze the features defined in an information template. In particular, we propose to analyze those features that present a complex nature. The nature of these features is described by an information template that is associated to another entity. In this case, there exists an explicit connection between the entity for which the complex nature feature is defined and the entity that describes the complex nature itself. Then, we

5. Requirements Traceability

consider that the classes derived from these two entities need to be related in the structural model. To capture this derivation Rule S6 is defined.

Rule S6 also has a weak traceability.

The Rule:

Rule S6:

TDE: Each *feature* that: (1) is associated to an entity A and (2) has a *complex nature*, which is described by the information template associated to an entity B

Maps to: an association relationship between the classes derived from the entities A and B.

Traceability: Weak

An Example: Figure 5.10 shows an example of how Rule S6 is applied. In this case, this rule derives an association relationship between the classes Item and Product from the *Product* feature associated to the entity Item.

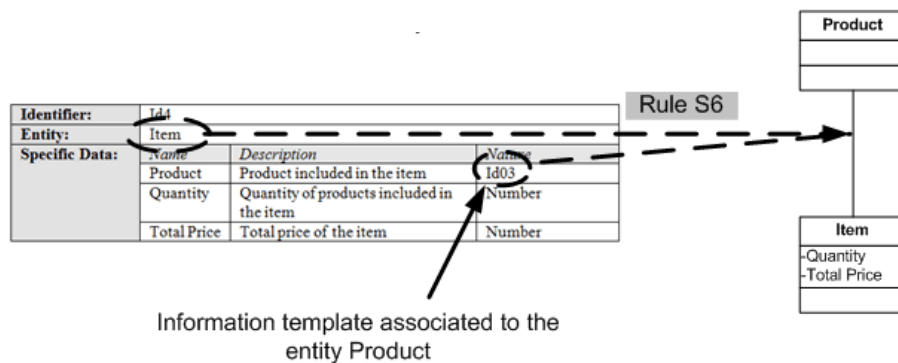


Figure 5.10 Examples of application of Rule S6

Rule S7: Super-Type Entities to Generalization Relationships

Rationale: At the conceptual level, a generalization relationship implies that a class (the specialized class) is based on another class (the general class). Roughly speaking, this means that an object of the specialized class is also an object of the general class. At the requirements level, this type of relations is captured in the definition of information templates. In particular, we can identify generalization relationships from those information templates associated to entities that are super-types of other entities (sub-types). In these cases, the specific data section that describes the super-type entity is not defined as a set of features but it is defined as an aggregation of the specific data sections that describe the sub-type entities. Then, information templates associated to super-types present an explicit relation between the super-type entity

5. Requirements Traceability

and the sub type entities. In these situations, we propose to define a generalization relationship between the class derived from the super-type entity and the classes derived from each sub-type entity. The class derived from the super-type entity constitutes the general class. The classes derived from the sub-type entities constitute the specific classes. This derivation is handled by Rule S7.

This rule has a weak traceability. This means that the identified generalization relationships can be substituted by an equivalent solution if analysts consider it adequate.

The Rule:

Rule S7:

TDE: Each entity which constitutes the sub-type of another entity (which is the *super-type*)

Maps to: a generalization relationship between the class derived from the entity super-type (general class) and the entity sub-type (specific class).

Traceability: weak

An Example: Figure 5.11 shows an example of how Rule S7 is applied. According to this figure, a generalization relationship is defined between the class product and the classes CD, Book and Software. This relationship is derived because the entity Product is a super-type of the entities CD, Book and Software.

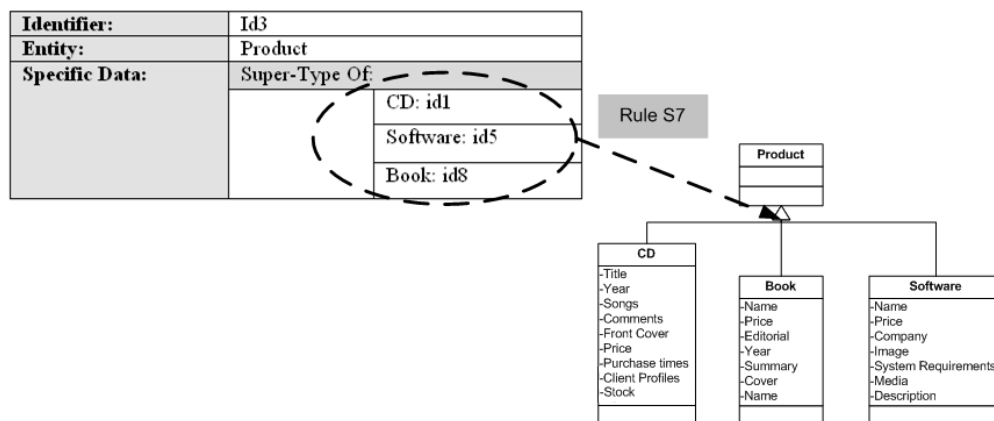


Figure 5.11 Example of application of Rule S7

3. Transaction Rules

Next, we introduce traceability rules that allow us to derive local transactions from task-based requirements models. Following the OOMethod approach (which is taken

5. Requirements Traceability

as base by the OOWS method), a local transaction is an execution unit that has a granularity higher than operations [Pastor et al. 2001]. A local transaction is made up of a set of operations that belong to a same class. Local transactions are part of a class definition (the class to which the operations belong). They are represented in the structural model in the same way than operations do (in the lower side of a class). In order to derive transactions, three rules are defined: Rule S8a, Rule S8b (which are strongly related to each other and then we analyze them together) and Rule S8c.

Rule S8a and S8b: Sequence of System Actions to OO-Method Local Transactions

Rationale: A local transaction is defined as a set of operations. We derive operations from function system actions (see Rule S3a). Then, local transactions are derived from a set of function system actions. In particular, we derive a transaction from sequences of function system actions whose first action is activated from an Output IP. The Output IP is used to identify the class in which the transaction is defined. This class is the class derived from the entity associated to the Output IP (by means of Rule S1). This derivation is captured by Rule S8b.

As explained above, local transactions are included in the definition of a class and they are defined from operations of this class. In this context, if a sequence of system action is derived into a transaction of a class, this means that each system action of the sequence has been previously derived into an operation of the class. However, only the first function system action has been derived into a class operation by means of Rule S3a. The rest of system actions (which are not directly connected to an Output IP) have been not derived into a class operation. To solve this, Rule 8a has been defined. This rule derives a class operation from each system action of the sequence except from the first one. The class in which operations are defined is the class derived from the entity associated to the Output IP from which the first action is activated.

These rules have a weak traceability. This means that the derived transactions are proposed elements. Software engineers can modify or delete transactions because they consider it properly for obtaining a better conceptual model.

The Rules:

Rule S8a:

TDE: Each *function system action* that: (1) is involved in a sequence of system actions, (2) it is not the first action of the sequence and (3) the first action is activated from an Output IP,

Maps to: a class operation, which is defined in the class derived from the entity associated to the Output IP.

Traceability: Weak

5. Requirements Traceability

Rule S8b:

TDE: Each sequence of function system actions, whose first action is activated from an Output IP,

Maps to: a local transaction, which is defined in the class derived from the entity associated to the Output IP.

Traceability: Weak

An Example: Figure 5.12 shows the application of Rule S8a and Rule S8b to the task Add CD. Rule S8b derives a transaction in the class CD from the sequence of system actions *Add_to_Cart* and *Update_Stock*. The operations that make up this transaction are those derived from both system actions. The first system action has been previously derived into the operation *Add_to_Cart()* of the class CD (by means of Rule S3a, see Figure 5.6). The operation that supports the second system action (*Update_Stock()*) is derived by Rule S8a. Notice how the name of the transaction is systematically defined from the string “Trans” plus the name of the first system action of the sequence.

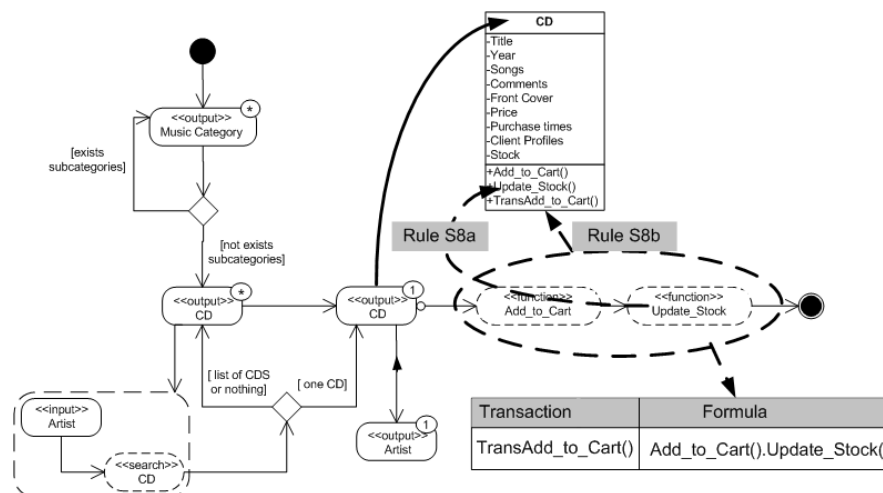


Figure 5.12 Example of application of Rule S8a and S8b

Rule S8c: Exchanged Data to Transaction Parameters

Rationale: Rule S8c is defined for deriving local transaction parameters. To do this, we follow an analogous strategy to the one presented for deriving operation parameter (see Rule S3b). In this case, we analyze the Inputs IPs that are associated to the different system actions from which the transaction is derived. For each feature defined in the data exchanged templates associated to these Input IPs, a transaction parameter is derived. This aspect is capture by Rule S8c.

5. Requirements Traceability

This rule has a weak traceability. This means that the derived transaction parameters are proposed elements. Software engineers can modify or delete them if they consider it properly for obtaining a better conceptual model.

The Rule:

Rule S8c:

TDE: Each feature defined in a *exchanged data template* which is associated to an Input IP that: (1) is connected to a function system action and (2) this action has been derived into a class operation which form part of a transaction

Maps to: a parameter of the transaction.

Traceability: Weak

An Example: This rule cannot be applied to the example presented above because none of the system actions that define the transaction are connected to an Input IP.

4. Rule Traceability Analysis

In this section, we introduce the structural model that is obtained when the set of rules introduced above are applied to the requirements model presented in the Chapter 4 (a partial version of Amazon Example requirements model, see Appendix D for a more complete version). We also identify in this section those elements of the structural model that cannot be systematically derived by applying the proposed traceability rules.

Obtained Structural Model

Figure 5.13 shows the structural model that has been derived from the requirements described in the previous chapter.

The classes CD, Music Category and Artist as well as the relationships between these classes are derived from: (1) the performance description of the task Add CD and (2) the information templates associated to the entities defined in this description. The derivation of these classes has been explained in detail throughout the current section.

The classes Item and Product are derived from the performance description of the task Inspect Shopping Cart. The corresponding information templates are also used. This derivation has been also introduced above.

The classes Order and Customer as well as their relationships are derived from the performance description of the task Checkout. A detailed explanation of this derivation can be found in Appendix D.

5. Requirements Traceability

The classes Book and Software have been included in order to properly illustrate the generalization relationship. They are derived from the tasks Add Book and Add Software. The rest of classes that derive from these tasks are not included in the structural model skeleton in order to not overload the figure. They are presented in Appendix D.

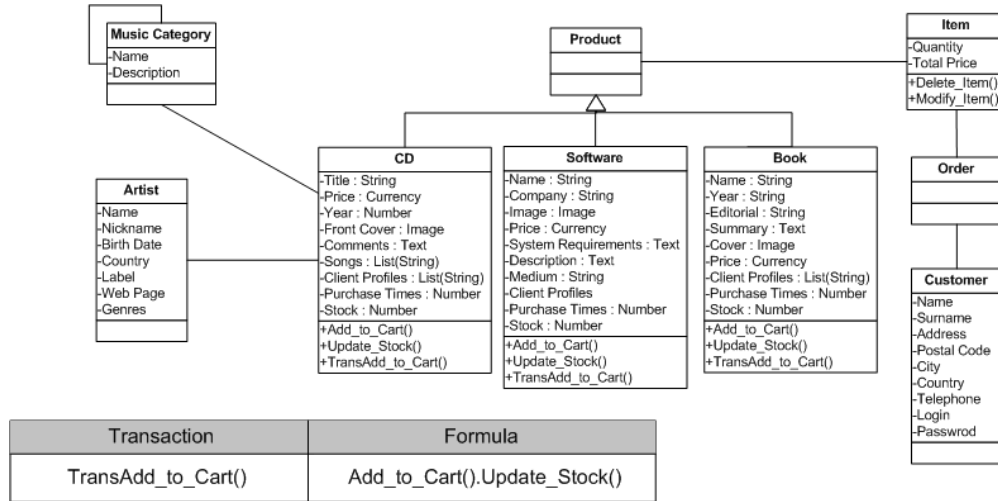


Figure 5.13 Partial version of the structural model of the Amazon example

Structural aspects that are not systematically derived

As stated at the introduction of this chapter, the presented traceability rules allow us to obtain a skeleton of the OOWS structural model. In order to obtain a complete structural model, analysts must manually refine and complete the obtained skeleton. To facilitate this task, we introduce next several aspects that cannot be systematically identified by the traceability rules:

- 1. Aggregation/Composition relationships.** Traceability rules only derive association relationships. These association relationships should be transformed many times into aggregation or composition relationships in order to properly capture the relationship semantics. For instance, in the structural model skeleton presented in Figure 5.13 an association relationship between the class Music Category and the class CD is defined. Analysts may consider that an aggregation relationship captures the semantic of the relationship between classes Music Category and CD in a better way than an association relationship.
- 2. Cardinality of relationships.** The cardinality of the different association relationships are not identified by means of the traceability rules. They must be manually defined by analysts in order to obtain a complete structural model.

5. Requirements Traceability

- 3. Generalization aspects.** Traceability rules allow us to detect generalization relationships between classes. However, depending on the way in which elementary tasks are described, operations and attributes that must be defined in the general class are separately identified for each specialized class. For instance, when traceability rules are applied to the Amazon example there are two operations (`Add_to_Cart()` and `Update_Stock()`) and one transaction (`TransAdd_to_Cart()`) that are defined in the classes `CD`, `Book` and `Software` (specialized classes in the class hierarchy). Analysts should define both the two operations and the transaction in the class `Product` (general class in the class hierarchy).
- 4. Transaction names.** Transactions are identified from sequence of system actions. However, the name that is assigned to each of this transaction is a systematic generated name (the string “Trans” plus the name of the first operation). Analysts should manually modify these names in order to properly adjust with the transaction semantics.
- 5. Function system actions as initial nodes.** Function system actions are derived into class operations either when they are activated from an Output IP (by means of Rule S3a) or when they are part of a transaction (by means of by Rule S8a). However, there is no rule that considers function system actions when they constitute the initial step of a task. There is no derivation in these cases because it is not able to systematically detect the class in which the operation must be created (because they are not connected to an Output IP, see Rule S3a and Rule S8a). Operations that support these function system actions must be manually defined by analysts.
- 6. Logic task operations are not supported.** Traceability rules presented above allow us to derive class operations from those system actions that can be activated from an Output IP because they are related to the IP entity (e.g. the system action *Add_to_Cart*, which is related to entity `CD`, see Figure 5.10). However, those system actions that must be activated from an Output IP in order to accomplish with the task logic are not properly supported in the structural model (e.g. system actions such as *identify_customer* or *create_customer*, see task Checkout in Figure 4.22). The problem in these cases is the same as the one explained for the point five: the class to which these operations belong cannot be systematically identified by traceability rules. This aspect must be manually defined by analysts. To do this, analysts can: (1) create an operation in an existent class in order to support the system action or (2) create a new class that incorporate an operation that support the system action. For instance, considering the task Checkout presented in Figure 4.22, analysts may consider properly to create a new operation in the class `customer` that supports the system action *create_customer*. As far as, the system action *identify_customer* it may be supported by a new class. For instance, analyst

5. Requirements Traceability

should decide to create a class *System* that provides operations for controlling meta-aspects (aspects that are not related to the Web application domain but they are related to the use of the Web application itself) such as the login and logout of users.

5.3.2 Navigational Model

In this section, we present a set of traceability rules that allows us to systematically derive the OOWS navigational model (see Appendix B) from the task-based requirements model. These rules are classified into two types (see Figure 5.14):

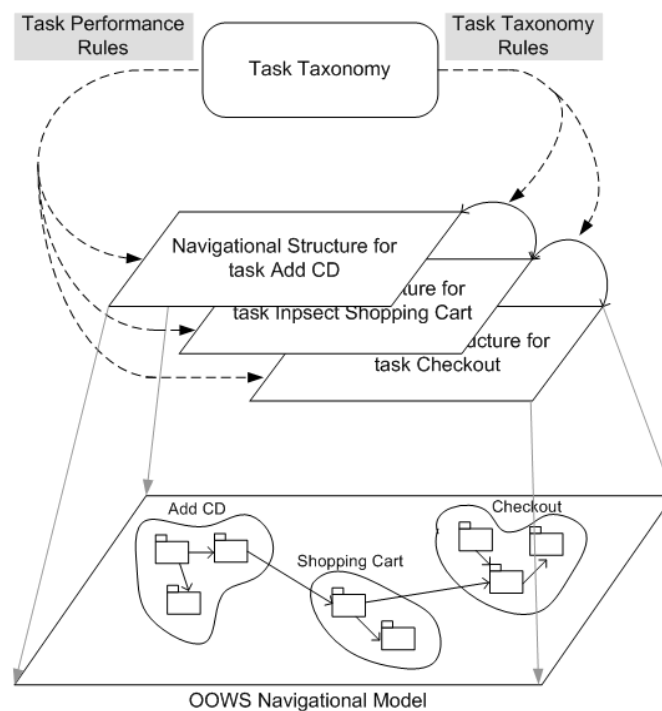


Figure 5.14 Derivation of the OOWS Navigational Model

1. Task Performance Rules: These rules are applied to the descriptions of the performance of each elementary task (activity diagrams). These rules allow us to derive a partial navigational structure which supports the performance of each elementary task at the conceptual level.
2. Task Taxonomy Rules: These rules are applied to the task taxonomy. These rules extract valuable navigational information from the temporal relationships defined among tasks. This navigational information is used to properly

5. Requirements Traceability

interconnect at the conceptual level the partial navigational structures derived from rules of type 1.

1. Task Performance Rules

Rules of this type are grouped in five categories according to the conceptual elements that are derived. We propose rules that derive:

- The user diagram (Rules N1a and N1b).
- Navigational contexts (Rule N2)
- The view of each navigational context (Rules N3a, N3b, N3c, N3d, N3e and N3f).
- Access mechanisms: indexes and search filters (Rules N4a, N4b, N4c, N4d, N4e and N4f).
- Navigational links (Rules N5a and N5b).

In order to explain each rule, we present first a brief rationale about the rule definition. Next, the rule itself is presented. Finally, an example of the rule application is shown.

Rule N1a: Type of Users to OOWS User Roles

Rationale: The OOWS user diagram represents the different types of user that can interact with the Web application by means of user roles (see Appendix B). At the requirements level, this information can be derived from the task templates that characterize elementary tasks.

Each of these templates characterizes an elementary task by means of four fields: goals, users, frequency and additional constraint (see Chapter 4, Figure 4.6). The *users* field indicates the users that can perform the task. This information can be used to define the user roles that must be included in the OOWS user diagram. This aspect is handled by Rule N1a. This rule has a strong traceability. This means that the Web application has to support the different derived user roles by imperative.

The Rule:

Rule N1a:

TDE: Each *type of user* defined in the field *users* of a task template

Maps to: a user of the user diagram.

Traceability: Strong

5. Requirements Traceability

An Example: Figure 5.15 shows how Rule N1a and Rule N1c are applied to the task template associated to the elementary task Add CD. In this example, Rule N1a creates a type of user for each user identified in the task template: Visitor and Customer.

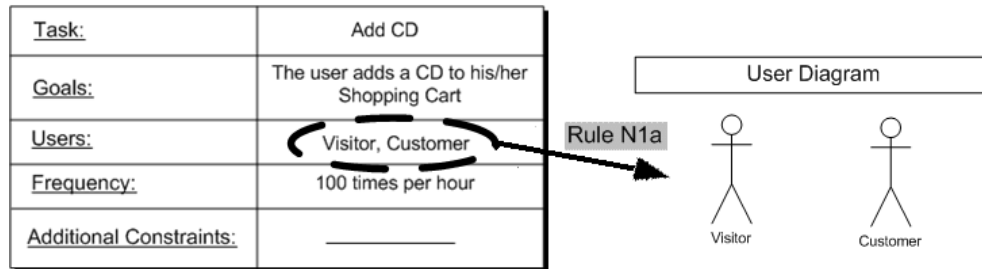


Figure 5.15 Example of application of Rule N1a

Rule N1b: Type of Users to Inheritance Relationships between User Roles

Rationale: The user diagram also allows us to define inheritance relationships among users that share navigation capabilities. The *users* field can also be used to derive inheritance relationships. To do this, we must identify the whole set of tasks that can perform each type of user. Then, by analyzing if a type of user can perform all the tasks allowed for another type of user, we can derive an inheritance relationship between both types of user. This aspect is precisely defined by Rule N1b.

This rule has a weak traceability. This means that inheritance relationships between user roles can be modified if software engineers consider it properly.

The Rule:

Rule N1b:

TDE: Each *type of user B* that: (1) can perform all the tasks allowed for a type of user A and (2) can perform additional tasks that are not allowed for the user A

Maps to: an inheritance relationship whose source is the user A and whose target is the user B.

Traceability: Weak

An Example: Rule N1b must be applied over the whole set of task templates in order to identify inheritance relationships. To properly illustrate the application of this rule we should present the whole set of task templates defined for the Amazon example. However, in order to not overload this example, we only apply this rule over two task templates. Figure 5.16 shows how a inheritance relationship has been defined between the user role Visitor and Customer because Customers can perform each of the tasks allowed for Visitors (Add CD) plus at least one more (Checkout).

5. Requirements Traceability

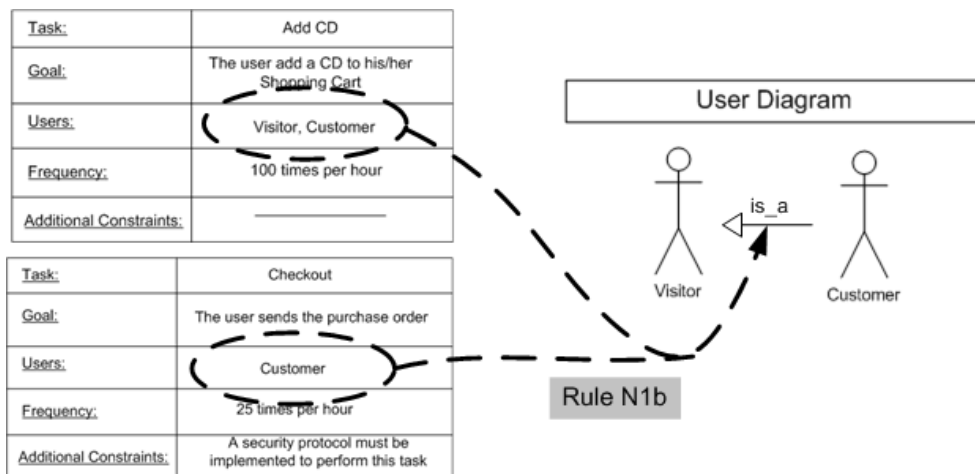


Figure 5.16 Example of application of Rule N1b

Rule N2: Output IPs to Navigational Contexts

Rationale: At the conceptual level, each navigational context describes specific information that is shown to users (see Appendix B). At the requirements level, an Output IP represents a step in the performance of an elementary task where the system provides the user with information. In this context, we derived navigational contexts from Output IPs defined in the different task performance descriptions. This derivation is described in detail by Rule N2.

As the *Except When* field of Rule N2 shows, not every Output IP derives into a navigational context. An Output IP does not derive into a navigational context when it provides information about multiple instances of an entity and also allows users to access another IP that provides information about only one instance of the same entity. These IPs represent a step in a task where the user compares elements of a list with each other (instances of the first IP) in order to select one and then obtain more detailed information about this instance (in the next Output IP). These situations are captured in the OOWS navigational model by means of the *index* primitive, which is handled by Rule N4a.

Rule N2 is defined with traceability strong. This means that the set of identified navigational contexts is mandatory for correctly supporting the performance of each task.

5. Requirements Traceability

The Rule:

Rule N2:

TDE: Each *output IP*

Maps to: a navigational context whose manager class is defined from the entity associated to the Output IP.

Except When: the IP: (1) informs about multiple instances (cardinality $*$) of one entity and (2) provides the user with access to another IP that informs about only one instance of the same entity (cardinality 1).

Traceability: Strong

Some Examples: Figure 5.17 shows how Rule N2 is applied to the elementary task Inspect Shopping Cart. According to this figure, two navigational contexts are derived from this task: (1) the navigational context Item that is derived from the IP Output(Item,*) and (2) the navigational context Product that is derived from the IP Output(Product,1).

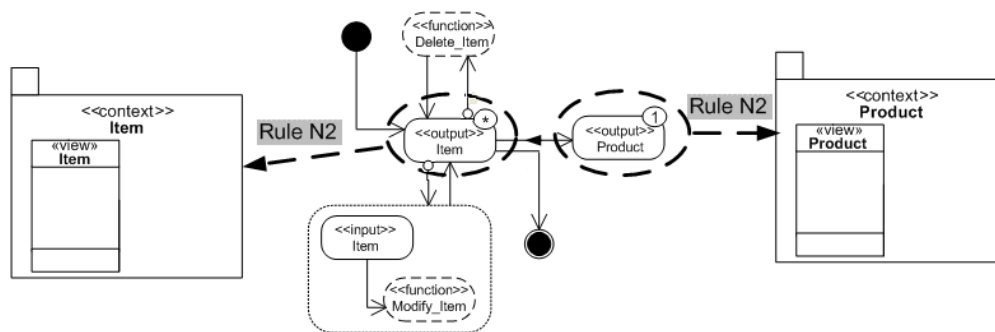


Figure 5.17 Example of application of Rule N2

Figure 5.18 shows another example of Rule N2 application. In this case, it is applied to the elementary task Add CD, from which three navigational contexts are obtained: *Music category*, *Artist* and *CD*. The context Music Category is derived from the IP Output(Music Category,*). The context CD is derived from the IP Output(CD,1). The context Artist is derived from the IP Output(Artist,1). No context is derived from the IP Output(CD,*) because it allows the user to access the IP Output(CD,1) (same entity, only one instance). The manager class of each context is derived from the entity of the IPs.

5. Requirements Traceability

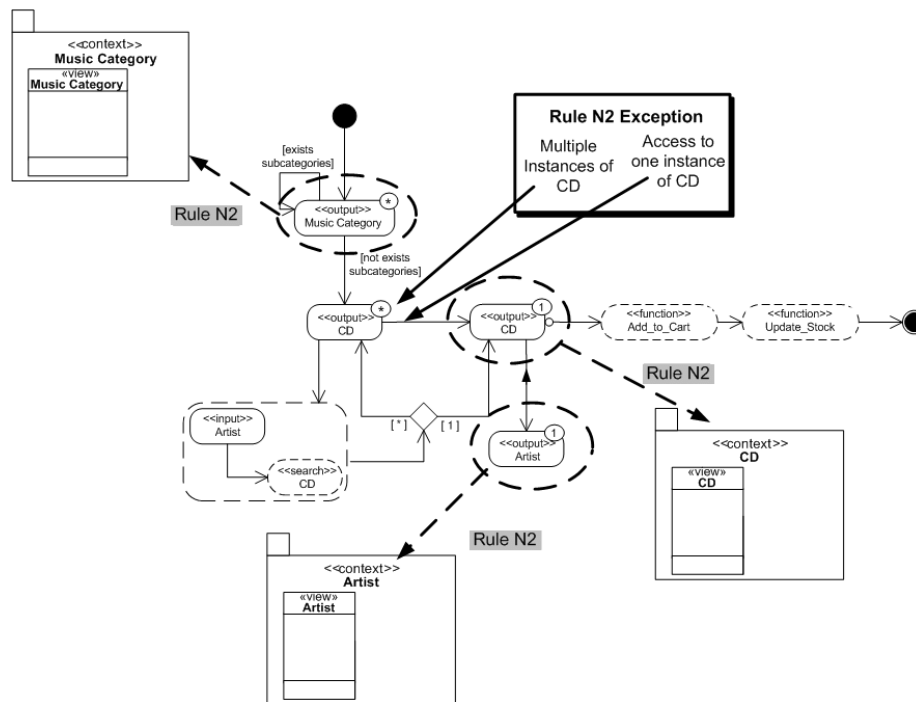


Figure 5.18 Another example of application of Rule N2

Rules N3a and N3b: Entity Features to Class Attributes and Complementary Classes

Rationale: The rule presented above allows us to derive the manager class of each navigational context view. However, neither class attributes nor complementary classes that extend the information provided by the manager class are derived. We can do this by analyzing exchanged data templates. These templates allow us to define the data that is provided to users in each Output IP. This data is defined by associating a set of entity features to the Output IPs. In the case that the Output has been derived into a navigational context, these features can be used to define manager class attributes and complementary classes. These derivations are described in detail by Rule N3a and Rule N3b.

Rule N3a and Rule N3b have both a weak traceability. The reason is that these rules obtain the attributes and complementary classes of the navigational context views. This information is shown in each navigational context and we consider it to be proposed information because analysts could modify, delete or add either new attributes or complementary classes without critically altering the performance of a task.

5. Requirements Traceability

The Rules:

Rule N3a:

TDE: Each *entity feature* that:

- (1) has a simple nature,
- (2) is associated (by means of an exchanged data template) to an Output IP that has been derived into a navigational context *C*,
- (3) is defined for the same entity as the one associated to the Output IP

Maps to: an attribute in the manager class of the navigational context *C*.

Traceability: Weak

Rule N3b:

TDE: Each *entity feature* that:

- (1) has a simple nature,
- (2) is associated (by means of an exchanged data template) to an Output IP that has been derived into a navigational context *C*,
- (3) is defined for an entity *E* that is different from the one associated to the Output IP

Maps to: (1) a complementary class in the navigational context *C*. It is defined over *E*. It is associated to the manager class of *C* by means of a context-dependency navigational relationship.
(2) An attribute in the complementary class. This attribute is defined from the entity feature name.

Traceability: Weak

An Example: Figure 5.19 shows the application of Rule N3a and Rule N3b to the task Add CD. In particular, we show the derivation performed from the features defined in the exchanged data template associated to the IP Output(CD,1). Rule N3a identifies manager class attributes from the features defined for the same entity as the one associated to the Output IP (entity CD). These features are: title, price, year, songs, comments and front cover. Rule N3b identifies complementary classes from the features that are defined for an entity (entity Artist) that is different from the entity associated to the Output IP (entity CD). In this case, there is only a feature: name of the artist. Then, this rule creates the complementary class Artist with the attribute name.

5. Requirements Traceability

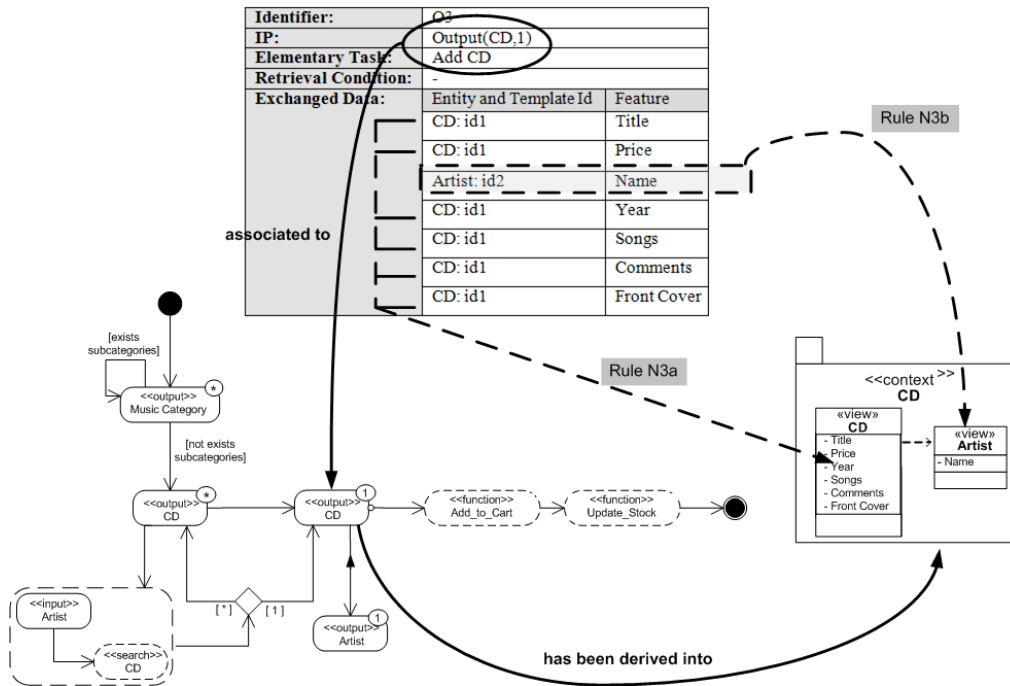


Figure 5.19 Example of application of Rules N3a and N3b

Rules N3c: Super-Type Entities to Complementary Classes

Rationale: A special case to be considered in the derivation of complementary classes is when an Output IP is associated to a super-type entity. In this case, the exchanged data template that is associated to the IP is defined from features that belong to the sub-type entities. The Output IP provides information about a sub-type entity of its associated entity (instead of information about the own associated entity). Let's consider for instance, the IP Output (Product, 1) defined in the task Inspect Shopping Cart (see Figure 4.24). According to the exchanged data template associated to this IP (see Figure 4.27), users access different information depending on the type of the product that must be shown. If the product is a CD, then the title, the price, the year, etc. are shown. If the product is a software product, then the name, the price, the company, etc. are shown. If the product is a book, then the name, the price, the editorial, etc. are shown.

In these cases, the retrieval of information that is defined in the corresponding navigational context constitutes a hierarchy of navigational classes. According to Rule N2, the navigational context derived from each Output IP has a manager navigational class derived from its associated entity. In the case introduced above, the IP Output (Product, 1) is derived into a navigational context whose manager class is Product. We complement this manager class with complementary classes that are

5. Requirements Traceability

derived from each sub-type entity (one for CD, one for Software and one for book). Complementary classes are connected to the manager class by means of context dependency relationships. According to the OOWS method (see Appendix B), these hierarchies of navigational classes retrieves the set of object that belongs to the manager class (in this case, Product). For each of these objects: the system shows first the attributes defined in the manager class; next, the system analyzes the sub-type of each object (in this case, CD, Software and Book); and finally, the system shows the attributes defined in the complementary class that matches its sub-type. The derivation of this hierarchy of classes is supported by Rule N3c.

Rule N3c has also a weak traceability.

The Rule:

Rule N3c:

TDE: Each exchanged data template that is associated to an Output IP which:

- (1) is associated to an entity *E1* that is a *super-type* of other entities
- (2) has been derived into a navigational context *C*,

Maps to: (1) a complementary class in the navigational context *C* for each entity that is a sub-type of *E1*. Each of these complementary classes is connected to the manager class by means of a context dependency relationship.
(2) Attributes in the complementary classes derived from the sub-type entity features that are defined in the exchanged data template.

Traceability: Weak

An Example: Figure 5.20 shows the application of the Rule N3c to the exchanged data template associated to the IP Output(Product,1). This example shows how the view of the context Product is defined. We associate the complementary classes CD, Software and Book to the manager class Product. Furthermore, we define the attributes of these complementary classes from the feature associated to the entities CD, Software and Book in the exchanged data template.

5. Requirements Traceability

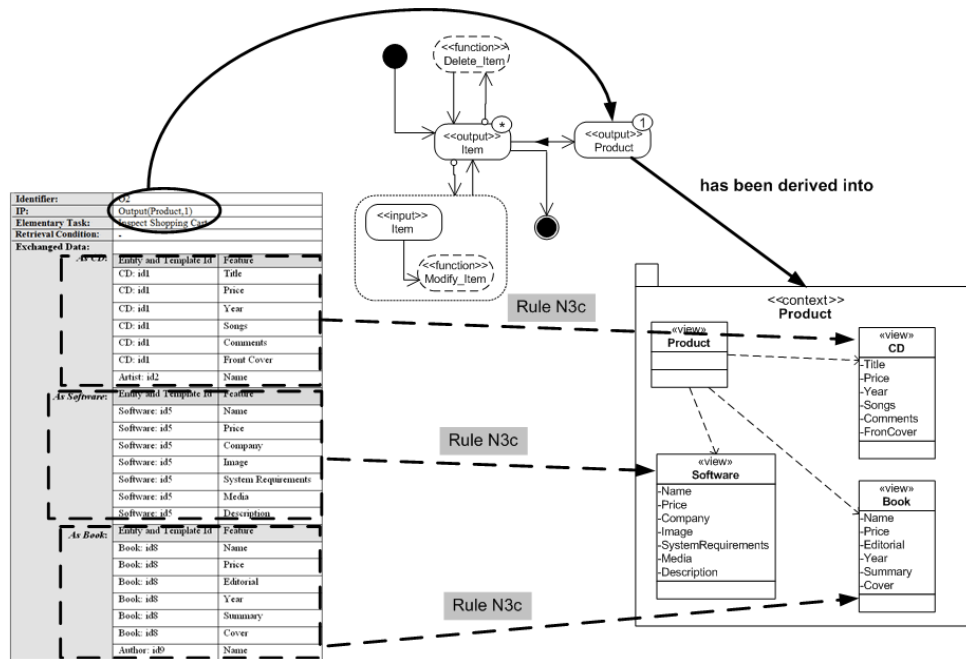


Figure 5.20 Example of application of Rules N3c

Rules N3d:

Complex Nature Features described by a Super-Type to Complementary Classes

Rationale: In order to properly derive complementary classes we must also take into account those exchanged data templates that (1) associate features with a complex nature to an Output IP and (2) the complex nature is defined by a super-type entity. An example of this is the IP Output(Item,*) that is defined in the task Inspect Shopping Cart (see Figure 4.18). This IP provides users with information about the items included in the shopping cart. For each item, the system provides users with the product associated to the item, the quantity of added products and the total price of the item. The first feature (the associated product) has a complex nature. The complex nature is described by the entity Product which is the super-type of the entities CD, Software and Book. Then, we indicate features of the sub-type entities in order to be shown in the IP Output(Item,*) when the product is a CD (we show the title of the CD, see Figure 4.28), a software product or a book (in this two cases we show the name, see Figure 4.28).

In these cases, the information retrieved by the corresponding navigational context is also defined as a hierarchy of navigational classes. As explained above, we must consider first that IPs such as Output(Item,*) are derived into navigational context whose manager class is defined from the IP entity (in this case, Item). We complement this manager class with the following complementary classes: (1) One

5. Requirements Traceability

complementary class defined over the super-type entity (in this case, Product) that is connected to the manager class of the context. (2) One complementary class for each sub-type entity (one for CD, one for Software and one for book). These complementary classes are connected to the complementary class defined over the super-type entity (Product). Rule N3d describes this type of derivation in detail.

Rule N3d has also a weak traceability.

The Rule:

Rule N3d:

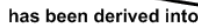
TDE: Each *feature* that has a *complex nature* and:

- (1) is associated to an Output IP that has been derived into a navigational context *C*,
- (2) the complex nature is described by the entity *E1*;
- (3) *E1* is the super-type of other entities

Maps to: (1) One complementary class in the navigational context *C* derived from *E1*; it is connected to the manager class of the context by means of a context-dependency navigational relationship;
(2) One complementary class for each sub-type entity; they are connected to the previous defined complementary class by means of a context dependency navigational relationship.
(3) Attributes for each complementary class derived from a sub-type entity. They are derived from the feature with complex nature.

Traceability: Weak

An Example: Figure 5.21 shows the application of rules N3a and N3d to the exchanged data template associated to the IP Output(Item,*). This example shows how the view of the context Item is defined. Rule N3a identifies the manager class attributes (quantity and total price). Rule N3d identifies the complementary classes (Product, CD, software and book) from the feature that must be shown in the IP and whose nature is Product.



5. Requirements Traceability

The Rule:

Rule N3e:

TDE: Each *function system action* that:

- (1) is activated from an Output IP that has been derived into a navigational context *C*,
- (2) is related to the Output IP entity,

Maps to: an operation of the manager class of the navigational context *C*.

Except When: the system action is connected to other system action and they have been derived into a local transaction in the structural model

Traceability: Strong

An Example: Figure 5.22 shows the application of Rule N3e to the performance description of the task Inspect Shopping Cart. This rule allows us to derive the operations *Delete_Item* and *Modify_Item*. These operations are defined in the manager class of the context *Item* because the corresponding system actions can be activated from the Output(*Item*,*) IP.

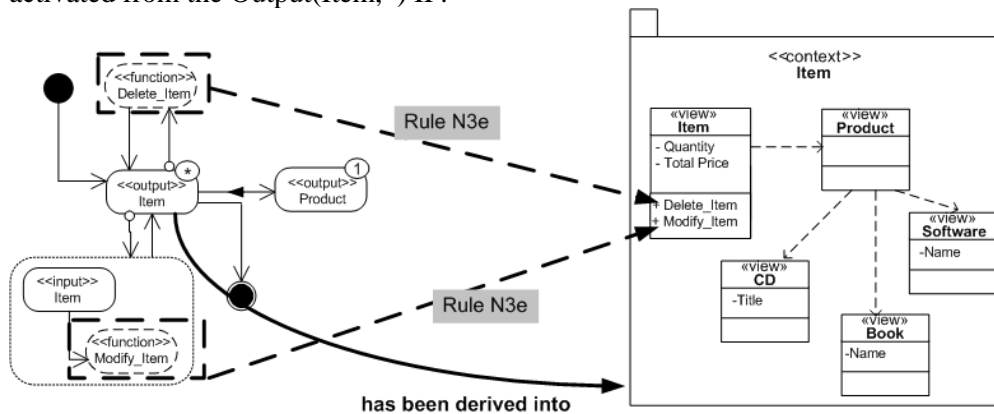


Figure 5.22 Example of application of Rule N3e

Rules N3f: Sequence of Function System Actions to Manager Class Operations

Rationale: System actions that are activated from an Output IP but are also involved in the definition of a local transaction are not handled by Rule N3e. In these situations, we must provide access to the whole transaction and not only to the first system action. To support this aspect Rule N3f is defined.

Rule N3f has a strong traceability because it is mandatory that navigational contexts provide users with access to the transaction in order to properly perform the task.

5. Requirements Traceability

The Rule:

Rule N3f:

TDE: Each *sequence of function system actions* that:

- (1) has been derived into a transaction T in the structural model,
- (2) the first action of the sequence is activated from an Output IP that has been derived into a navigational context C,
- (3) the first action is related to the Output IP entity,

Maps to: a transaction defined in the manager class of the navigational context C.

Traceability: Strong

An example: Figure 5.23 shows the application of Rule N3f to the description of the performance of the task Add_CD. By means of this rule the transaction Trans1() that is derived from the sequence of actions *Add_to_Cart*, *Update_Stock()* (see Rule S8a, Figure 5.12) is defined in the manager class of the context CD. Notice how the transaction is defined in this context because the *Add_to_Cart* system action (the first action of the sequence) can be activated from the Output(CD,1) IP (IP from which the context is derived).

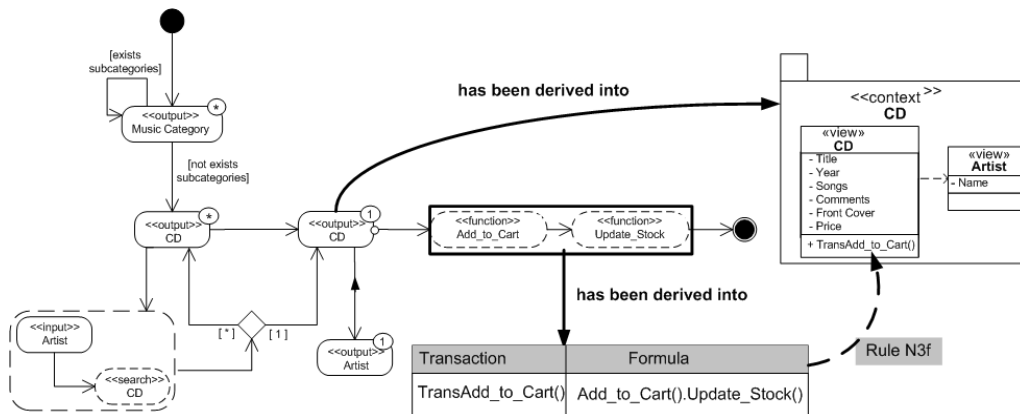


Figure 5.23 Example of application of Rule N3f

Rules N4a and N4b: Output IPs to Indexes

Rationale: Some Output IPs represent steps during the performance of a task where the user compares elements of a list with each other in order to select the desired one. These IPs are those that both inform about multiple instances of one entity (cardinality *) and provide the user with access to a second IP that informs about only one instance of the same entity (cardinality 1). In the OOWS navigational model, these list of elements that are used to facilitate the access to a specific one are modelled by means of the *index* primitive. Thus, as Rule N4a states, each Output IP that accomplishes the conditions mentioned above are derived into an *index*. The

5. Requirements Traceability

activation mode of the index must be *always*. This means that the index will be automatically activated before obtaining the information defined in the navigational context (see Appendix B). Furthermore, according to Rule N4b, index attributes are defined from the entity features that are associated to the Output IP by means of an exchanged data template.

Rule N4a and Rule N4b have a weak traceability because software engineers may decide to support this list by means of other conceptual elements instead of an index. For instance, software engineers may consider a better solution to model this list of elements by means of another navigational context in order to obtain particular navigational characteristics.

The Rules:

Rule N4a

TDE: Each Output IP that:

- (1) informs about multiple instances (cardinality *) of one entity,
- (2) provides the user with access to a second IP that informs about only one instance of the same entity (cardinality 1)

Maps to: an index that is attached to the navigational context derived from the second IP. The activation mode of this index is *always*.

Traceability: Weak

Rule N4b

TDE: Each entity feature associated to an Output IP which has been derived into an index

Maps to: an attribute of the index.

Traceability: Weak

An Example: Figure 5.24 shows how an index is derived from an Output IP. In this example, Rule N4a and Rule N4b are applied to the elementary task Add CD. Rule 4a derives an index from the IP Output(CD,*) IP. This index is attached to the navigational context derived from the IP Output(CD,1) because both IPs are connected. The attributes of the index are derived by means of Rule N4b from the exchanged data template associated to the IP Output(CD,*).

5. Requirements Traceability

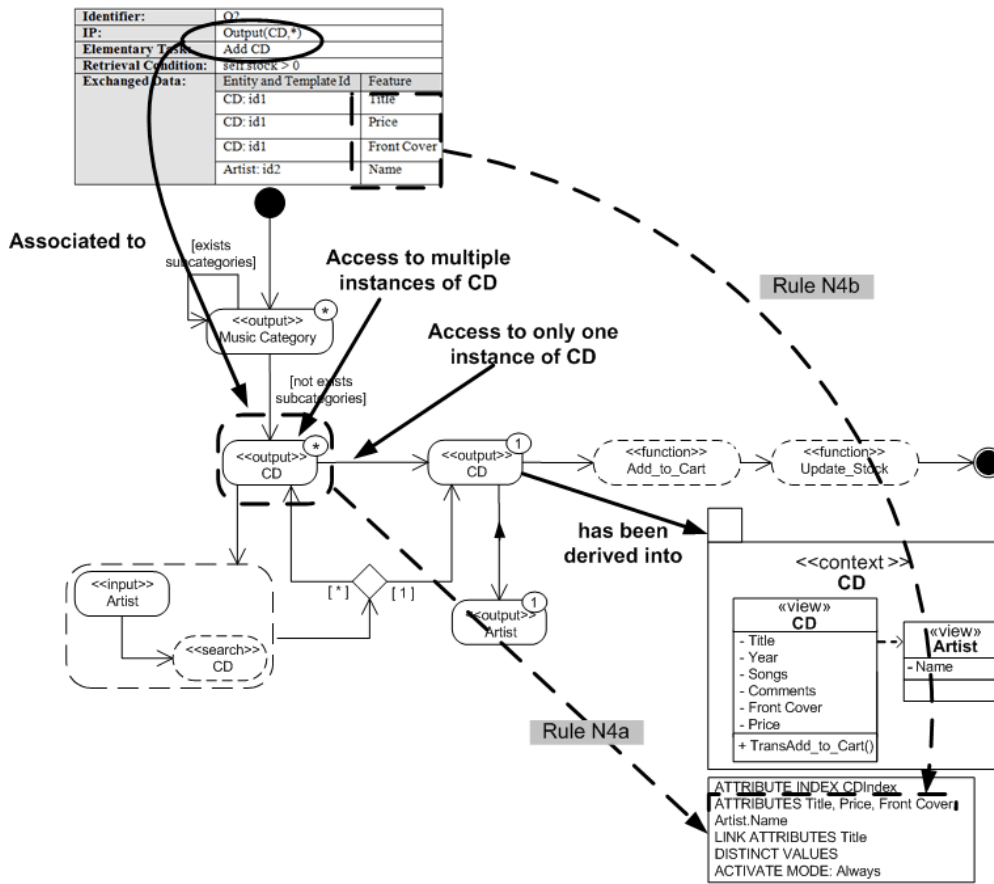


Figure 5.24 Example of application of Rules N4a and N4b

Rules N4c and N4d: Search System Actions to Search Filters (with activation “never”)

Rationale: Search system actions perform an inquiry over the system state in order to obtain specific information. This kind of queries can be supported at the conceptual level by the OOWS search filter primitive. We can derive this primitive from the search system actions that are activated from an Output IP and moreover they inquiry the system in order to obtain information about the same entity associated to the Output IP.

These situations are supported at conceptual level by defining a search filter which is attached to the navigational context derived from the Output IP. The activation mode of the search filter must be *never*. This means that the search filter must be explicitly activated by the user (see Appendix B). In the case that the Output IP derives into an index instead of a navigational context the search filter is attached to the navigational context to which the index is attached.

5. Requirements Traceability

In order to capture these derivations both Rule 4c and 4d are defined. These rules have a strong traceability. Users need to perform the searches in order to properly satisfy the task logic. In the OOWS navigational model, a search filter is the only primitive that allow us to represent these searches at the conceptual level. Then, the derived search filters are mandatory because no other conceptual elements can replace them.

The Rules:

Rule N4c

TDE: Each *search system action* that is activated from an Output IP and:

- (1) the output IP has been derived into a navigational context and
- (2) the search system action inquiries information related to the same entity associated to the Output IP.

Maps to: A search filter that is defined in the navigational context. The activation mode of the search filter is *never*.

Traceability: Strong

Rule N4d

TDE: Each *search system action* that is activated from an Output IP and:

- (1) the output IP has been derived into an index and
- (2) the search system action inquiries information related to the same entity associated to the Output IP.

Maps to: A search filter that is defined in the navigational context to which the index is attached. The activation mode of the search filter is *never*.

Traceability: Strong

An example: In order to better illustrate how these rules are applied to a task performance description we present an example where Rule 4Nd is applied together with Rule N4g (which derives search filter attributes). This example can be further found in the explanation of Rule N4g (see Figure 4.25).

Rules N4e and N4f: Search System Actions to Search Filters (with activation “always”)

Rationale: In order to properly derive OOWS search filters we must also to analyse those search system actions that constitute the initial step in a task performance description. In these situations, the results obtained by the search system action are shown to the user by means of an Output IP (then, the system action is connected by means of an arc to an Output IP).

These situations are supported at the conceptual level by defining a search filter that is attached to the navigational context derived from the Output IP. The activation mode of the search filter must be *always*. This means that the search filter will be

5. Requirements Traceability

automatically activated when the user access the navigational context (see Appendix B). In the case that the Output IP derives into an index instead of a navigational context the search filter is attached to the navigational context to which the index is attached. In order to capture these derivations both Rule N4e and N4f are defined.

Rule N4e and N4f have a strong traceability.

The Rules:

Rule N4e

TDE: Each *search system action* that constitutes the initial step of a task and:

- (1) it is connected to an Output IP and
- (2) the Output IP has been derived into a navigational context

Maps to: A search filter that is defined in the navigational context. The activation mode of the search filter is always.

Traceability: Strong

Rule N4f

TDE: Each *search system action* that constitutes the initial step of a task and:

- (1) it is connected to an Output IP and
- (2) the Output IP has been derived into an index

Maps to: A search filter that is defined in the navigational context to which the index is attached. The activation mode of the search filter is *always*.

Traceability: Strong

An example: There are no situations in the Amazon Example where these rules can be applied. They are however applied in a analogous way to the Rule N4d (for which an example is shown in Figure 4.25).

Rule N4g: Exchanged Data to Search Filter Attributes

Rationale: Rules N4c, Rule N4d, Rule N4e and Rule N4f allow us to derive search filters from search system actions. However, the attributes that define these filters are not derived. In order to derive these attributes from a task performance description we must consider that search system actions inquiry the system state according to a specific criterion. This criterion is introduced by users throughout Input IPs. Thus, in the situations presented above, search filter attributes can be derived from the entity features that are associated to the Input IP that allows users to introduce the search criterion. To capture this derivation Rule N4g is defined.

Rule N4g has a weak traceability. We consider the search criterion to be a proposed element that software engineer could modify without altering in a critical way the correct performance of a task. For instance, software engineers can decide to search CDs from a title instead of from an artist's name.

5. Requirements Traceability

The Rule:

Rule N4g

TDE: Each entity feature that is associated to an Input IP (by means of an *exchanged data* template) that allows users to introduce the search criterion of a *search* system action

Maps to: an attribute of the filter derived from the *search* system action.

Traceability: Weak

An Example: Figure 5.25 shows an example where a search filter is derived from a search system action. Rule N4d and Rule N4g are applied to the elementary task Add CD. Rule N4d derives a search filter from the search system action Search(CD). This search system action is activated from the IP Output(CD,*). Since this IP is derived into an index (see Figure 5.24), the search filter is attached to the navigational context to which the index is attached. Attributes of the search filter are derived by means of Rule N4g from the exchanged data template associated to the IP Input(Artist, Search(CD)).

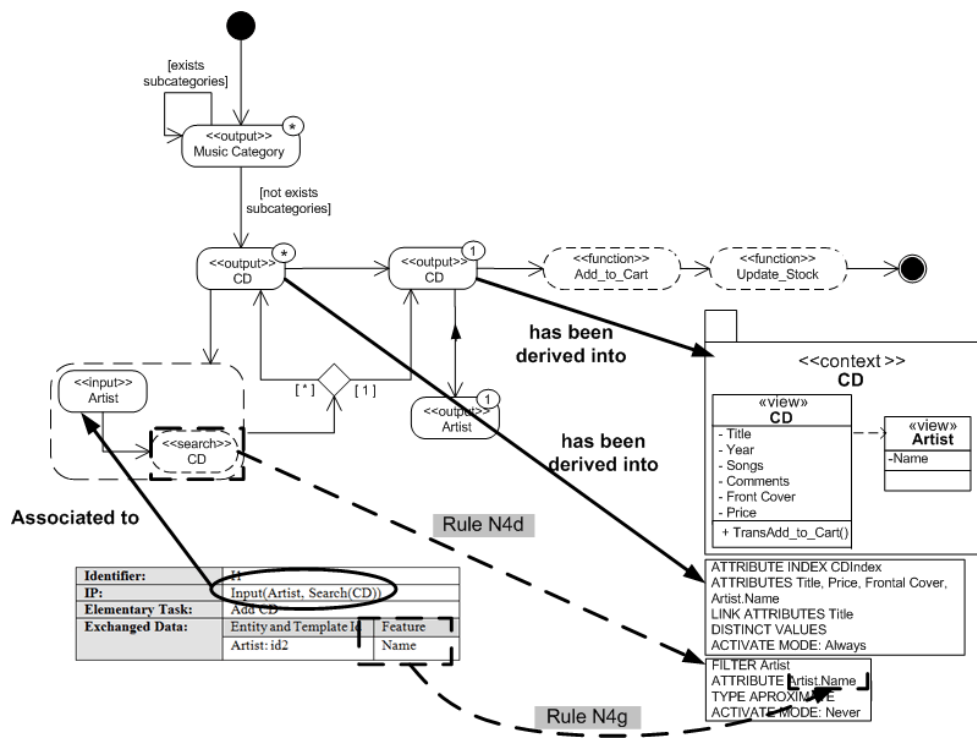


Figure 5.25 Example of application of Rules N4d and N4g

5. Requirements Traceability

Rule N5a and N5b: Output IP Connections to Sequence Navigational Links

Rationale: The OOWS method proposes two kind of navigational links (see Appendix B): (1) Exploration links and (2) Sequence Links. The first ones are detected by analyzing the temporal relationships defined among tasks in the task taxonomy (further explained in detail). The second ones are derived by analyzing the performance descriptions of elementary tasks. This second aspect is explained next.

Previously to introduce traceability rules, it is worth to remark that the definition of a sequence navigational link implies the definition of context relationship in the navigational context that constitutes the link's source, or vice versa (see Appendix B). Thus, the rules that we present next derive both sequence navigational links and context navigational relationships at the same time.

Taking into account the description of an elementary task performance, navigational links are derived from connections between Outputs IPs. These connections (defined by means of arcs in the activity diagram) define the sequence in which the user must access each Output IP in order to perform the task. Output IPa represent steps in a task where the system provides users with some information. We know that these steps are supported at the conceptual level by means of navigational contexts (see Rule N2). Then, in order to preserve the sequence of Outputs IPs defined at the requirement level we must define the corresponding sequence of navigational contexts at the conceptual level. This sequence of navigational contexts is defined by connecting them throughout navigational links. Thus, we define a navigational link between two navigational contexts if the IPs from which the contexts have been derived are: (1) connected by means of an arc or (2) connected through an Output IP that has not derived into a context. To detect this, we define Rules N5a and N5b.

Rules N5a and N5b have a strong traceability. This means that users must be able to activate the derived navigational links in order to satisfy the task logic.

The Rules:

Rule N5a

TDE: Two output IPs that derive into two different navigational contexts and both IPs are directly connected by means of an arc

Maps to: a navigational link between both navigational contexts.

Traceability: Strong

Rule N5b

TDE: Two output IPs that derive into two different navigational contexts and both IPs are connected through another IP that has not been derived into a context

Maps to: a navigational link between both navigational contexts.

Traceability: Strong

5. Requirements Traceability

An Example: Figure 5.26 shows how Rule N5a and Rule N5b are applied to the task Add CD. Rule N5b derives a navigational link between the contexts Music Category and CD. This link is defined because the IP Output(Music Category,*) and the IP Output(CD,1) (IPs from which the contexts are derived) are connected through the IP Output(CD,*) (which has not derived into any context). Rule N5a derives a navigational link between the contexts CD and Artist. This link is defined because the IP Output(CD,1) and the IP Output(Artist,1) are connected by means of an arc.

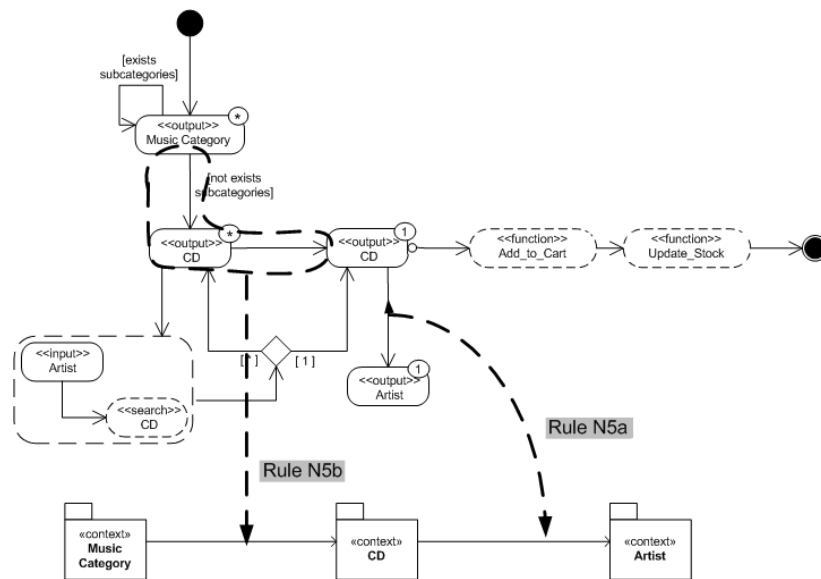


Figure 5.26 Example of application of Rules N5a and N5b

Each derived navigational links implicitly defines a context navigational relationship in the source context (see Appendix B). These navigational relationships are defined by connecting a navigational class of the source navigational context to a complementary class. This complementary class must be the same as the manager class of the target navigational context. Then, we need to create this complementary class if it does not exist. Figure 5.27 shows the navigational relationships defined in both navigational contexts Music Category and CD (since they are the source context of each derived navigational link). We can see in the context Music Category how the manager class is connected to the complementary class CD (manager class of the target context, CD; it is indicated between brackets under the navigational relationship). This complementary class is newly created in order to define the context navigational relationship. We can also see how the manager class in the context CD is connected to the complementary class Artist (which is the manager class of the target context Artist). In this case, a context-dependency navigational relationship (those that only retrieve information, without navigation) between the classes CD and Artist has been already defined in the context CD by means of Rule

5. Requirements Traceability

N3b (see Figure 5.19). Thus, we can see in Figure 5.27 how this navigational relationship has been transformed into a context navigational relationship.

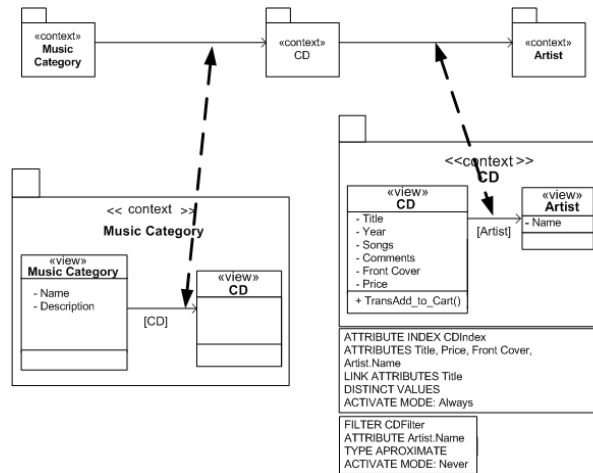


Figure 5.27 Implicitly defined navigational relationships

2. Task Taxonomy Rules

In this section, we present a set of traceability rules that allows us to derive navigational information from the task taxonomy defined at the requirements level. We know that each elementary task is derived into a navigational structure that is made up of a set of navigational contexts together with a set of navigational links (see rules presented above). These contexts and links are the abstract mechanisms that support each task at the conceptual level. We also know that a temporal relationship defined between two tasks indicates a coordinated way in which both tasks must be performed. This coordination indicates aspects such as which task must be performed before, if one task can be interrupted by another task and so on (see temporal relationships in Section 4.2.2). This coordination can be used to derive navigational information that allows us to properly relate the different partial navigational structures at the conceptual level. For instance, the temporal relationships defined in the task taxonomy of the Amazon example indicate us that users must always be able to inspect the shopping cart while they are adding products. From this information we can derive that the navigational structure that supports the adding of products must be extended with mechanisms to make the shopping cart accessible to users wherever.

Next, we analyze the coordination between tasks that represent each temporal relationship in detail. We indicate the navigational information that can be extracted from them and how it is supported in the OOWS navigational model. However, to correctly understand this analysis we must present some previous considerations.

5. Requirements Traceability

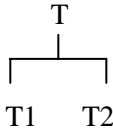
Previous Considerations

Consideration 1: We must consider that the coordination proposed by tasks is not always fully supported by the conceptual primitives proposed by the OOWS method. Thus, each temporal relationship is analyzed by means of the following fields:

- *Coordination:* This field explains the coordination between tasks that represents the temporal relationship.
- *OOWS support:* In this field, we indicate which part of the coordination can be supported by the OOWS method. We also indicate the OOWS conceptual primitives that must be derived in order to support the coordination. The traceability rules that support these derivations are also presented.
- *Unsupported characteristics:* In this field, we indicate which part of the coordination between tasks cannot be supported by the conceptual primitives proposed by the OOWS method.
- *Example:* If the rule is applicable to the Amazon example, we show the OOWS conceptual elements that are derived when the rule is applied.

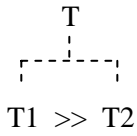
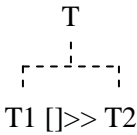
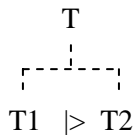
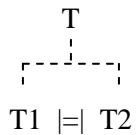
Consideration 2: In order to analyse how temporal relationships affect the navigational structure we need to know the initial and final step of each task. To know the initial and final step of elementary tasks is trivial (we just need to analyze its associated performance description). However, temporal relationships can be defined between non-elementary tasks. In order to identify the initial and final step of these non-elementary tasks we need to consider: (1) the initial and final step of its subtasks, (2) the type of refinement used to obtain the subtasks and (3) the semantics of the temporal relationship defined between subtasks, if a temporal refinement is used. We present next a table where these aspects are studied²⁰.

Table 5.1 Study of initial and final task steps from temporal relationships

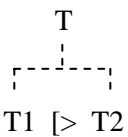
Situation	Analysis
	<p><u>Description:</u> The task T is decomposed into the subtasks T1 and T2 by means of a structural refinement.</p> <p><u>Identified Steps:</u> In order to perform T we must perform either T1 or T2. Thus, T can start and finish in two different ways. Then, we consider that the initial and final steps of T are either the initial and final steps of T1 or the initial and</p>

²⁰ This analysis only takes into account tasks that are refined into two sub-tasks. The study of more complex refinements (that is, tasks that are refined into three or more subtasks) are left as further work.

5. Requirements Traceability

	<p>final steps of T2.</p> <p>Initial (T) = Initial(T1) and Final(T)= Final(T1) xor Initial (T) = Initial(T2) and Final (T) = Final (T2)</p>
	<p><u>Description:</u> The task T is decomposed into the subtasks T1 and T2 by means of a temporal refinement. The temporal relationship used is <i>Enabling</i>.</p> <p><u>Identified Steps:</u> In order to perform the task T the subtask T1 must be first performed and next the subtask T2. Then, we consider that the initial step of T is the initial step of T1 and the final step of T is the final step of T2.</p> <p>Initial (T) = Initial (T1) Final (T) = Final (T2)</p>
	<p><u>Description:</u> The task T is decomposed into the subtasks T1 and T2 by means of a temporal refinement. The temporal relationship used is <i>Enabling with information passing</i>.</p> <p><u>Identified Steps:</u> In order to perform the task T the subtask T1 must be first performed and next the subtask T2. Then, we consider that the initial step of T is the initial step of T1 and the final step of T is the final step of T2.</p> <p>Initial (T) = Initial (T1) Final (T) = Final (T2)</p>
	<p><u>Description:</u> The task T is decomposed into the subtasks T1 and T2 by means of a temporal refinement. The temporal relationship used is <i>Suspend/Resume</i>.</p> <p><u>Identified Steps:</u> In order to perform T the subtask T1 must be performed. During the performing of T1 it can be interrupted by T2. When T2 is finished T1 is resumed. Then, we consider that the initial and final steps of T are the initial and final steps of T1.</p> <p>Initial (T) = Initial (T1) Final (T) = Final (T1)</p>
	<p><u>Description:</u> The task T is decomposed into the subtasks T1 and T2 by means of a temporal refinement. The temporal relationship used is <i>Task independence</i>.</p> <p><u>Identified Steps:</u> In order to perform T we must perform both T1 and T2. However, we can perform first T1 and then T2 or</p>

5. Requirements Traceability

	<p>we can perform first T2 and then T1. Thus, we consider that the initial and final steps of T are both the initial and final steps of T1 and the initial and final steps of T2.</p> <p>Initial (T) = Initial(T1) and Final(T)= Final(T1) xor Initial (T) = Initial(T2) and Final (T) = Final (T2)</p>
 <p>T ┌-----┐ └-----┘ T1 > T2</p>	<p><u>Description:</u> The task T is decomposed into the subtasks T1 and T2 by means of a temporal refinement. The temporal relationship used is <i>Disabling</i>.</p> <p><u>Identified Steps:</u> In order to perform T we must begin to perform T1. Then, this subtask is interrupted by T2. When T2 finishes T is finished. Thus, we consider that the initial step of T is the initial step of T1 and the final step of T is the final step of T2</p> <p>Initial (T) = Initial (T1) Final (T) = Final (T2)</p>

Consideration 3: Finally, in order to derive the proper conceptual mechanisms from temporal relationships we need to know the navigational contexts that support the initial and final step of each task. These navigational contexts change depending on the type of node that defines the initial and final step:

Initial step. The initial step of a task can be: an Output IP, a search system action or a function system action (see Chapter 4):

Output IPs: In this case, the navigational context that supports the initial step is either the navigational context derived from the Output IP (in this case that Rule N2 is applied) or the navigational context to which the index derived from the Output IP is attached (if Rule N4a is applied).

Search system actions: In this case, the navigational context that supports the initial step is the navigational context to which the filter derived from the action (by means of Rule N4b) is attached.

Function system actions: When the initial step of a task is a function system action no conceptual elements are derived (see rules presented above). Thus, no navigational information can be extracted from these situations and then they are not considered in the analysis.

5. Requirements Traceability

Final step. The final step of a task can be: an Output IP or a function system action (see Chapter 4).

Output IPs: In this case, the navigational context that supports the final step is the navigational context derived from the Output IP when Rule N2 is applied.

Function system actions: In this case, the navigational context that supports the final step is the navigational context in whose manager class the operation derived from the action is defined (see Rule N3e).

Finally, we also need to identify the navigational structure (the set of navigational contexts and navigational links) that supports a task. In the case of elementary tasks, this navigational structure is made up of the navigational contexts and links that are derived by applying the rules presented above. In the case of non-elementary tasks, this navigational structure is made up of the navigational structures that support their subtasks.

Temporal Relationship Analysis

Next, we present the analysis of each temporal relationship presented in the Section 4.2.2.

These rules have all a weak traceability. These rules define mechanisms to interconnect the partial navigational structures obtained from the rules presented above. However, these mechanisms are not the only that properly support this interconnection. Then, we allow software engineers to change them if they consider it opportune.

Enabling (>>)

Coordination: If a task T1 is related with another task T2 by using this temporal relationship it means that the task T2 must be performed after the task T1. Considering the navigational structures derived from both tasks, this relationship indicates that the navigational structure of T2 must be accessible from the navigational structure of T1 when users finish T1. In particular, the navigational context which supports the final step of T1 (hereafter known as C1) must provide access to the navigational context which supports the initial step of T2 (hereafter known as C2).

OOWS Support: In order to allow users to access C1 from C2 we need to consider two scenarios depending on the final step of T1.

Scenario 1: The final step of T1 is an Output IP. This scenario is not properly supported by the OOWS method. We explain the reasons below.

5. Requirements Traceability

Scenario 2: The final step of T1 is a function system action. The task finishes when the system performs the action. At the conceptual level, this system action is derived into an operation in the manager class of C1. Then, the task T1 is finished when users activate this operation. In this context, we must allow users to access C2 after the execution of this operation.

In order to better understand this scenario, let's consider for instance an E-Commerce application that provides users with the list of shopping cart items every time that a product is added. In this case, users add a product to the shopping cart by activating the operation *add_to_cart()* in a context *Product Description*. Furthermore, the list of shopping cart items is provided in a context *Shopping Cart*. Then, when users add a product from the context *Product Description* (by activating the operation *add_to_cart()*) we must allow users to automatically access the context *Shopping Cart*.

In order to support these situations, we must define a non-contextual service link (see Appendix B) in C1 (context CD Description in the above example). This service link is attached to the operation derived from the final step of T1 (operation *add_product()*). The context attribute of this service link is C2 (context Shopping Cart). Thus, by means of this service link, users automatically access C2 when the operation derived from the final step of T1 is executed. In this case, no information passing is needed since the service link is non-contextual. This aspect is captured by Rule N6s.

Rule N6a

TDE: Each *Enabling* relationship that is defined between two tasks T1 and T2, where:

- (1) C1 is the navigational context that supports the final step of T1,
- (2) C2 is the navigational context that supports the initial step of T2,
- (3) The final step of T1 is a function system action that has been derived into an operation O of the manager class of C1

Maps to: a non-contextual service link that is attached to the operation O. The context attribute of this link is C2.

Except When: T1 or a subtask of T1 present an iterative unary relationship.

Traceability: Weak

Rule N6a is not applied when T1 or one of its subtasks present an iterative unary relationship. When this unary relationship is defined, users are able to perform T1 or a subtask of T1 so many times as they want. In this context, we should not force users to start to perform T2 after performing the system action (by automatically redirecting them to the context that support the initial step of T2) because they may have not decided to finish performing the iterative task yet. In this case, we must allow users to start perform T2 when they want. To do this we define an exploration link that

5. Requirements Traceability

provides access to the navigational context that supports the initial step of T2. This derivation is supported by Rule N6b.

Rule N6b

TDE: Each *Enabling* relationship that is defined between two tasks T1 and T2, where:

- (1) T1 or a subtask of T1 presents an iterative unary relationship,
- (2) C2 is the navigational context that supports the initial step of T2,
- (3) The final step of T1 is a function system action

Maps to: an exploration link that provides access to C2.

Traceability: Weak

Unsupported characteristics: As introduced above, scenario 1 is not properly supported at the conceptual model by means of the abstract primitives proposed by the OOWS method.

According to this scenario, both the final step of T1 and the initial step of T2 are Output IPs. In this way, users finish T1 by analyzing the information that is provided by an Output IP. At the conceptual level, this Output IP is supported by a navigational context C1. Then, users finish T1 by analyzing the information provided by C1. This means that T1 is explicitly finished by users, i.e. the task will finish when users finish of analyzing the information. On the other side, users start to perform T2 by analyzing the information provided in the Output IP that constitutes its first step. At the conceptual level, this Output IP is supported by a navigational context C2. Then, users start to perform T2 by analyzing the information provided by C2. To support the semantics of the relationship, we need to define a mechanism that allows users to access C2 from C1 when they finish of analyzing the information of C1.

In order to better understand this scenario, let's consider for instance an E-learning application where users can study a specific subject. To do this, users must first study a set of contents (which are provided in a navigational context *Contents*) and then they must perform a set of exercises (which are provided in a navigational context *Exercices*). In this sense, we must allow users to access the context *Exercices* from the context *Contents* when users decide that have studied enough.

We can support this access by defining a sequence navigational link between both contexts. However, this type of links implies carrying contextual information to the target context and, according to the temporal relationship semantics (*enabling*, T2 must be performed after T1), C2 must be access form C1 without passing of information (the passing of information is captured by the temporal relationship *enabling with information passing*). Thus, the sequence navigational links do not allow us to support at the conceptual level the semantics of the temporal relationship *enabling*.

5. Requirements Traceability

The other type of navigational link that is proposed by the OOWS method is the exploration navigational link. This link provides access to a context without carrying information. However, this access is available for every context defined in the navigational model. According to semantics of the temporal relationship *Enabling*, we need to provide access to C2 from C1, but only from C1. Thus, exploration navigational links neither allows us to support the semantics of the temporal relationship *enabling* at the conceptual level.

Thus, the OOWS method does not provide us with a conceptual primitive that allows us to define an access relationship between two specific contexts without passing of information.

Enabling with information passing ([>>])

Coordination: This coordination is the same as the one presented above: If a task T1 is related with another task T2 by using an *Enabling with information passing* temporal relationship it means that the task T2 must be performed after task T1. However, in this case, T1 passes information to T2.

Considering the navigational structures derived from both tasks, this relationship indicates that the navigational structure of T2 must be accessible from the navigational structure of T1 when users finish T1. In particular, the navigational context that supports the final step of T1 (hereafter known as C1) must provide access to the navigational context that supports the initial step of T2 (hereafter known as C2). Furthermore, specific information must be carried from C1 to C2.

OOWS Support: In order to allow users to access C1 from C2 we need to consider again two scenarios, depending on the final step of T1.

Scenario 1: The final step of T1 is an Output IP. As we have explained above, users finish the task by analyzing the information provided by the navigational context C1. Thus, we must provide users with mechanisms that allow them to access C2. In this case, this mechanism must allow a passing of information between C1 and C2.

In order to better understand this scenario, let's consider for instance an E-Commerce application that provides users with information about CDs. This application allows users to query information about CDs with the special characteristic that when user accesses the description of a CD they can put on sale a used copy of the CD. In this case, users access first the description of a CD (which is provided in a navigational context *CD Description*) and then users can put the CD on sale by accessing a navigational context *Sale CD*. In this sense, we must allow users to access the context *Sale CD* from the context *CD Description*. In this access the CD to put on sale must be carried from the context *CD Description* to the context *Sale CD*.

5. Requirements Traceability

We can support this access with information passing at the conceptual level by means of a sequence navigational link defined between both contexts. This sequence navigational link allows users to access C2 (navigational context that support the initial step of T1) from C1 (navigational context that support the final step of T1). Furthermore, this navigational link implies carrying information from C1 to C2 fitting, in this way, the semantics of the temporal relationship enabling with information passing.

As we can see in Appendix B, a sequence navigational link is implicitly defined by a context navigational relationship. Thus, a context navigational relationship must be defined in C1. The target navigational class of this relationship must be the same as the manager class of the context C2. The context attribute of the relationship must be C2. This aspect is captured by Rule N7a.

Rule N7a

TDE: Each *Enabling with information passing* relationship that is defined between two tasks T1 and T2, where:

- (1) C1 is the navigational context that supports the final step of T1,
- (2) C2 is the navigational context that supports the initial step of T2,
- (3) the final step of T1 is an Output IP

Maps to: a context navigational relationship in C1 whose target navigational class is the same as the manager class of C2. The context attribute of the context navigational relationship is C2.

Traceability: Weak

Scenario 2: The final step of T1 is a function system action. This is the same case as the scenario 2 of the Enabling temporal relationship. We allow users access C2 from C1 by defining a service link in C1. However, an information passing must be supported in this case. To do this, the service link is defined as contextual. This means that objects over which the operation is executed are carried from C1 to C2. This aspect is captured by Rule N7b.

Rule N7b

TDE: Each *Enabling with information passing* relationship defined between two tasks T1 and T2, where:

- (1) C1 is the navigational context that supports the final step of T1;
- (2) C2 is the navigational context that supports the initial step of T2;
- (3) The final step of T1 is a function system action that is derived into an operation O of the manager class of C1.

Maps to: a contextual service link that is attached to the operation O. The context attribute of this link is C2.

Except: when T1 or a subtask of T1 present an iterative unary relationship.

Traceability: Weak

5. Requirements Traceability

In this same way as happens in the previous temporal relationship, Rule N7b is not applied if a unary temporal relationship is associated to T1 or a subtask of T1. These situations are supported by Rule N7c.

Rule N7c

TDE: Each *Enabling with information passing* relationship defined between two tasks T1 and T2, where:

- (1) T1 or a subtask of T1 presents an iterative unary relationship;
- (2) C2 is the navigational context which support the initial step of T2;
- (3) The final step of T1 is a function system action

Maps to: an exploration link that provides access to C2.

Traceability: Weak

Unsupported Characteristics: none.

Example: In the Amazon example, an *Enabling with information passing* relationship is defined between the tasks Collect Products and Checkout. According to the analysis presented above, the final step of Collect Products is defined by the final step of one of its subtasks Add CD, Add Software and Add Book. This final step is a function system action (see for instance the performance description of task Add CD Figure 5.2). However, the subtask Add Product presents an iterative unary relationship. Considering this situation, the navigational context that support the first step of the task Checkout is defined as an exploration navigational context by applying the Rule N7c. This derivation is graphically shown in Figure 5.28.

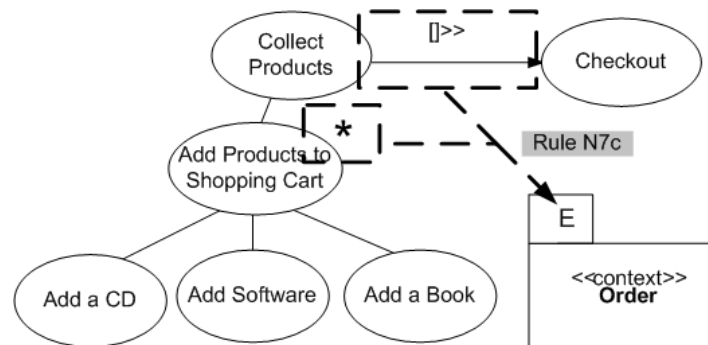


Figure 5.28 Example of application of Rule N7c

Suspend/Resume (|>)

Coordination: If a *suspend/resume* relationship is defined between two tasks, T1 and T2, it means that the user can interrupts T1 at any time to perform T2. Furthermore, when T2 is finished users must continue T1 in the point where it was suspended.

5. Requirements Traceability

Considering the navigational structures derived from both tasks, this relationship indicates that the navigational structure of T2 must be accessible from the any part of the navigational structure of T1. That is, the navigational context that supports the initial step of T2 must be accessible from any navigational context of the T1 navigational structure. Furthermore, the navigational context that supports the last step of T2 must provide access to the navigational context of T1 from which users suspend this tasks and start to perform T2.

OOWS Support: The OOWS conceptual primitives that we can use to support this coordination are the following: exploration links and subsystems. We connect an exploration link to the navigational context that supports the initial step of T2. This makes this navigational context accessible from any other navigational context defined in the navigational model. However, we only want this context to be accessible from navigational contexts derived from T1. To obtain this, we include navigational contexts derived from both tasks (T1 and T2) into a subsystem. A subsystem is a conceptual primitive that allows us to group navigational contexts in order to cope with complex navigational models (see appendix B). If there are no other navigational contexts derived from other tasks the navigational subsystem can be omitted. This derivation is captured by Rule N7a.

Rule N8

TDE: Each *Suspend/Resume* relationship defined between two tasks T1 and T2, where:

- (1) C is the navigational context that supports the initial step of T2;
- (2) CS_1 is the set of navigational contexts derived from T1;
- (3) CS_2 is the set of navigational contexts derived from T2.

Maps to: (1) an exploration navigational link which provides access to C and (2) a subsystem which includes navigational contexts in CS_1 and CS_2 .

Traceability: Weak

Unsupported Characteristics: The resuming aspect.

We have defined OOWS conceptual mechanisms that allow users to start T2 from any navigational context that supports the performance of T1. However, we cannot define mechanisms that allow users to access the navigational context where T1 was suspended once T2 is finished. To do this we would have to support two aspects:

- (1) Storing in run time the navigational context of T1 from which the user suspend T1.
- (2) Defining a navigational link that accesses this navigational context (the one stored in run time) from the navigational context that supports the last step of T2.

5. Requirements Traceability

Unfortunately, the OOWS method does not provide us with conceptual primitives that allow us to capture these two aspects at the conceptual level.

Example: In the Amazon example, a *Suspend/Resume* relationship is defined between the tasks Add Product to the Shopping Cart and Inspect Shopping Cart. According to the analysis presented above, the navigational structure that supports the task Add Product to the Shopping Cart is made up of the navigational structures that support its subtasks Add CD, Add Software and Add Book (that is, the set of navigational context and navigational links derived from each subtask). Figure 5.29 shows how the *Suspend/Resume* temporal relationship is supported at the conceptual level by applying Rule N8. In this figure, only the navigational contexts derived from the task Add CD are shown. The navigational context derived from the tasks Add Software and Add Book are omitted in order to not overload the example. We can see how Rule N8 connects an exploration relationship to the navigational context Item (the context that supports the initial step of Inspect Shopping Cart) and moreover includes the contexts derived from the tasks Add CD and Inspect Shopping Cart in a subsystem.

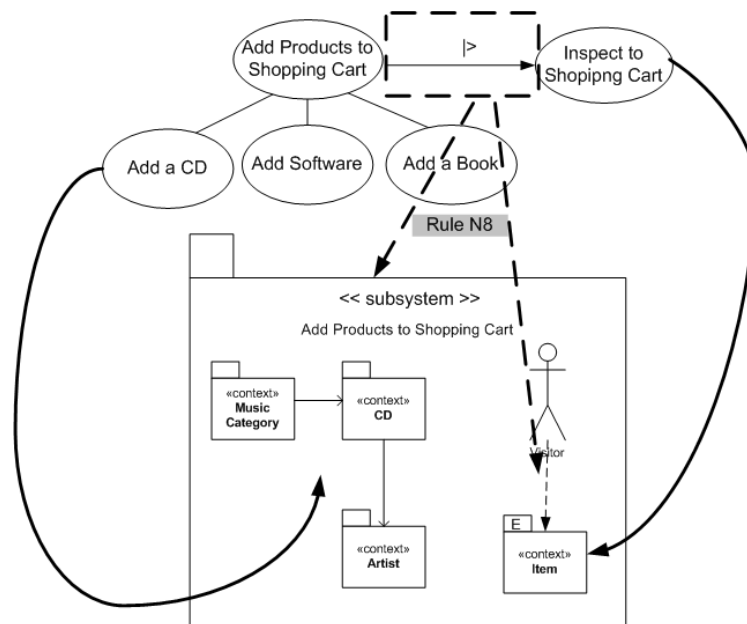


Figure 5.29 Example of application of Rule N8

Task Independence (\neq)

Coordination: If a *task independence* relationship is defined between two tasks, T1 and T2, it means that both tasks must be performed. However, the order in which they

5. Requirements Traceability

are performed is irrelevant. Thus, the user can either perform first T1 and then T2 or perform first T2 and then T1.

Considering the navigational structure derived from both tasks, this relationship indicates that: (1) users must initially be able to access both navigational structures in order to start to perform one of the tasks. And (2) once users start to perform a task the navigational structure of the other task must not be accessible until the current task is finished. That is, if users start to perform T1 (by accessing the navigational structure that supports this task) the navigational structure that support T2 must not be accessible until T1 is finished; and vice versa. In order to better understand this coordination let's consider for instance a Web application for supporting the work of teachers. This application can be used by teachers to evaluate the performance of students. To do this, teachers must both analyze the results obtained in the written exam and analyze the work performed by student during the whole year. We represent these two activities as two tasks: *Analyze Exams* and *Analyze Work*. In order to evaluate the performance of students, teachers must perform both tasks but the order in which these tasks are performed is not critical.

OOWS Support: The OOWS conceptual primitive that we can use to support this coordination is the exploration link. By (1) connecting an exploration link to the navigational context that supports the initial step of T1 and (2) connecting another exploration link to the navigational context that supports the initial step of T2, we are allowing users to initially access to the navigational structures that support both tasks. This aspect is captured by Rule N9.

Rule N9

TDE: Each *task independence* relationship defined between two tasks T1 and T2, where:

- (1) C1 is the navigational context that supports the initial step of T1;
- (2) C2 is the navigational context that supports the initial step of T2;

Maps to: (1) an exploration navigational link which provides access to C1 and (2) another exploration navigational link which provides access to C2.

Traceability: Weak

Unsupported Characteristics: Conditioned accessibility.

According to this temporal relationship, once users access a navigational structure in order to perform a task (e.g. T1), the navigational structure that supports the other task (e.g. T2) must not be accessible. However, this navigational structure must be accessible again when T1 is finished (that is, it must be accessible from the navigational context that supports the last step of T1). In this context, the solution to support this aspect at the conceptual level is to define an exploration link whose accessibility depends on a specific condition. For instance, a condition based on the

5. Requirements Traceability

navigation performed by users: if users access a specific navigational context (this one that supports the first step of T1) then the exploration link (this one that provides access to the navigational context that supports the first step of T2) is not accessible. However, this type of specifications is not supported by the OOWS method.

Disabling (\lhd)

Coordination: If a *Disabling* relationship is defined between two tasks, T1 and T2, it means that T1 is completely interrupted by the user in order to perform T2. This is the same situation as when a Suspend/Resume Relationship is defined between two tasks (see above) except from the fact that it is no necessary to define mechanisms that allow users to resume T1. Thus, considering the navigational structures derived from both tasks, this relationship indicates that the navigational structure of T2 must be accessible from any part of the navigational structure of T1. That is, the navigational context that supports the initial step of T2 must be accessible from any navigational context of the T1 navigational structure.

OOWS Support: The OOWS conceptual primitives that we can use to support this coordination are the same used in the Suspend/Resume relationship: exploration links and subsystems. We connect an exploration link to the navigational context that supports the initial step of T2. This makes this navigational context accessible from any other navigational context defined in the navigational model. However, we only want this context to be accessible from navigational contexts derived from T1. To obtain this, we include navigational contexts derived from both tasks (T1 and T2) into a subsystem. Thus, the navigational structure of T2 is accessible from any part of the navigational structure of T1. If there are no navigational contexts derived from other tasks the navigational subsystem can be omitted. This derivation is captured by Rule N10.

Rule N10

TDE: Each *Disabling* relationship defined between two tasks T1 and T2, where:

- (1) C is the navigational context that supports the initial step of T2;
- (2) CS_1 is the set of navigational contexts derived from T1;
- (3) CS_2 is the set of navigational contexts derived from T2.

Maps to: (1) an exploration navigational link that provides access to C and (2) a subsystem that includes navigational contexts in CS_1 and CS_2 .

Traceability: Weak

Unsupported Characteristics: none.

5. Requirements Traceability

Iterative Task (*)

Coordination: If a task T1 is defined as an *iterative task*, it means that users must be able to perform T1(again) once they have finished it. Considering the navigational structure that supports the task, this means that users must be able to access the navigational context that supports the initial step of T1 from the navigational context that supports the final step of T1.

OOWS Support: As we have explained above, the OOWS method does not allow us to connect two specific navigational contexts without passing of information. Thus, the conceptual primitive that we can use to support this temporal relationship is the exploration link. By connecting an exploration link to the navigational context that supports the initial step of T1 we allow users to perform this task so many times they need (however, as we explain below this solution does not support the relationship semantics in the proper way). The creation of the exploration link is defined in Rule N11.

Rule N11

TDE: Each defined *iterative task*, where:

- (1) C is the navigational context that supports the initial step of the task.

Maps to: an exploration navigational link which provides access to C.

Traceability: Weak

Unsupported Characteristics: The solution proposed above allows users to perform the task so many times they need. However, the possibility of starting to perform the task is always available during the whole performance of the task, not only at the end of the task. This aspect provides users with the possibility of stop iterative tasks at any point of its performance in order to start them again.

Example: In the Amazon example, the task Add Product to Shopping Cart is an iterative task. Since this task is not an elementary task, the iteration is supported at the conceptual level by defining the proper mechanisms in the navigational structure that support the elementary tasks obtained from it. For instance, we can see in Figure 5.30 how the navigational context that supports the initial step of the task Add CD is connected to an exploration link. This allows users to add a CD to the shopping cart so many times they need.

5. Requirements Traceability

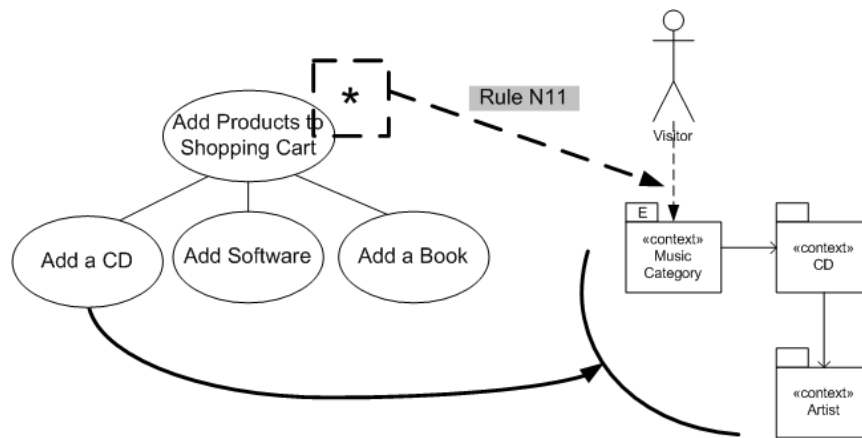


Figure 5.30 Example of application of Rule N11

Finite Iteration ((n))

Coordination: If a finite iteration is attached to a task T1, it means that users must perform the task the number of times that is indicated. This relationship is similar to the iterative one. Considering the navigational structure that supports the task, the finite iteration relationship indicates that users must be able to access the navigational context that supports the initial step of T1 from the navigational context that supports the final step of T1. However, the number of times that users can perform the task must be controlled.

OOWS Support: As happens with the iterative relationship we can only support this temporal relationship by defining exploration links. We connect an exploration link to the navigational context that supports the initial step of T1 and then we allow users to perform the task so many times they need. The creation of this exploration link is defined in Rule N12.

Rule N12

TDE: Each *finite iteration* relationship that is attached to a task, where:

- (1) C is the navigational context that supports the initial step of the task.

Maps to: an exploration navigational link which provides access to C.

Traceability: Weak

Unsupported Characteristics: The aspect that OOWS does not support is the same as the one explained in the iterative tasks: the definition of a mechanism that allows users to start to perform the task again but only from the navigational context that supports the last step of the task. Furthermore, a mechanism that allows us to indicate the number of times that the user can start the tasks cannot be indicated at the

5. Requirements Traceability

conceptual level. A solution to support this aspect could be a navigational link that: (1) allows us to connect two specific navigational contexts without information passing and (2) allows us to define accessibility conditions.

Optional Task ([])

Coordination: If a task is defined as optional users are not forced to perform it. That is, users are able to decide on whether or not the task must be performed. This unary relationship alone does not provide us with information that can be used to extend the navigational structure that supports the task. However, if we consider this unary relationship in combination with other binary relationship, the optional aspect may modify the way in which the binary relationship are supported at the conceptual level. In particular, an optional unary relationship modifies the way in which the two following binary relationships are supported: (1) Enabling relationship and (2) Enabling with information passing relationship.

If some of these two temporal relationships are defined between two tasks T1 and T2, it indicates that users can only perform T2 after finishing T1. However, if T1 (the source task) is defined as optional we must not restrict the access to T2 (the target task) only after T1 finishes (because users may decide on not performing T1). We must define a mechanism that allows users to start to perform T2 although T1 is not performed.

OOWS Support: In order to provide a mechanism that allows users to start T2 although T1 is not performed we define an exploration link. This exploration link provides access to the navigational context that supports the initial step of T2. This aspect is captured by Rule N13a and Rule N13b.

Rule N13a

TDE: Each *optional task* that is the source of an enabling relationship, where:

- (1) C is the navigational context that supports the initial step of the target task.

Maps to: an exploration navigational link that provides access to C.

Traceability: Weak

Rule N13b

TDE: Each *optional task* that is the source of a enabling with information passing relationship, where:

- (1) C is the navigational context that supports the initial step of the target task.

Maps to: an exploration navigational link that provides access to C.

Traceability: Weak

5. Requirements Traceability

A Further Traceability Rule

Rule N14: On the one hand, task performance rules derive a partial navigational structure that supports the performance of each elementary task. On the other hand, rules presented above analyze temporal relationships in order to define OOWS conceptual mechanisms (exploration navigational links, subsystems, and sequence navigational links) that allow users to properly access these navigational structures.

Every temporal relationship connects a source task to a target task. The traceability rules presented above indicate how user may access the navigational structure of the target task from the navigational structure of the source task according to the temporal relationship semantics. In this context, users must previously access the navigational structure of the source task in order to access the navigational structure of the target task. However, traceability rules do not define any conceptual mechanism that allows users to access the navigational structure of the source task. If a task is never defined as the target of a temporal relationship no mechanism is defined for allowing users to access its navigational structure.

For instance, Figure 5.28 shows the conceptual mechanisms that are defined to access the navigational structure of the task Inspect Shopping Cart from the navigational structure of the task Add Product to Shopping Cart (an exploration link and a subsystem). As we can see, these mechanisms are defined to allow users to access the navigational structure of the target task from the navigational structure of the source task. No conceptual mechanisms are defined to allow users to access the navigational structure of tasks such as Add CD or Add Product to Shopping Cart, which never constitutes the target task of a temporal relationship.

In order to define mechanisms that allow users to start to perform these tasks, we have defined Rule N14. This rule creates an exploration link that provides access to the navigational context that supports the initial step of tasks that never constitute the target of a temporal relationship.

Rule N14

TDE: Each task that never constitutes the target of a temporal relationship, where:
(1) C is the navigational context that supports the initial step of the task.

Maps to: an exploration navigational link that provides access to C.

Traceability: Weak

5. Requirements Traceability

Rule Traceability Analysis

In this section, we introduce the navigational model that is obtained when the set of rules presented above are applied to the requirements model introduced in the previous chapter (a partial version of Amazon Example requirements model). A full description of this case study can be found in Appendix D. We also identify in this section those elements of the navigational model that cannot be systematically derived by the proposed traceability rules.

Obtained Navigational Model

Figure 5.31 and Figure 5.32 show the navigational model that supports the requirements described in the previous chapter.

Figure 31 shows the navigational map defined for visitors. According to the navigational contexts defined in this map, visitors can access to information related to music categories, CDs and artists as well as to information related to items and products.

The navigational contexts Music Category, CD and Artist as well as the navigational links between these contexts are derived from the task Add CD. The navigational contexts Item and Product as well as the navigational link between them are derived from the task Inspect Shopping Cart. The reachability of each navigational context (Exploration/Sequence) is derived from the temporal relationships defined in the task taxonomy. Figure 5.32 shows the definition of the views of these contexts.

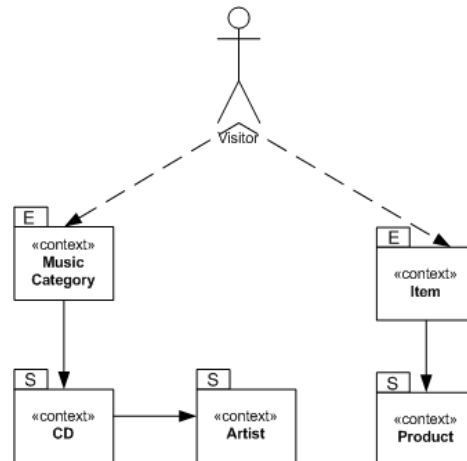


Figure 5.31 Partial version of the navigational map of the Amazon example

5. Requirements Traceability

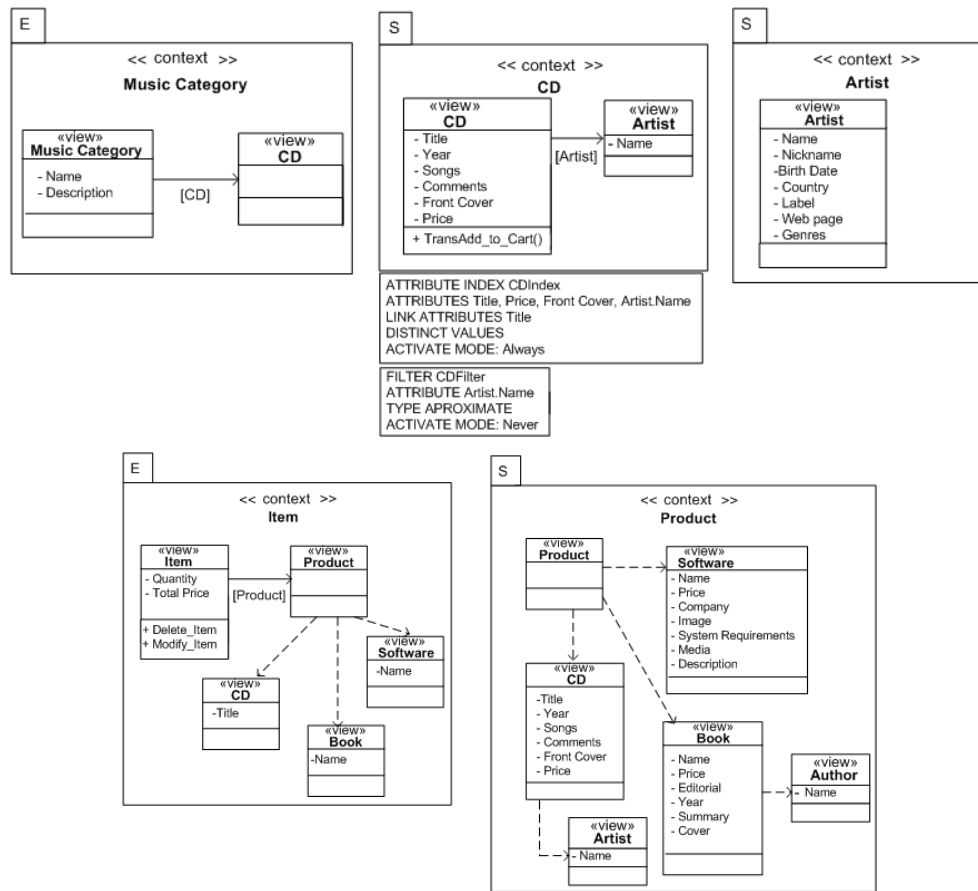


Figure 5.32 Partial version of the navigational contexts of the Amazon example

Navigational aspects that are not systematically derived

The presented traceability rules allow us to obtain a skeleton of the OOWS navigational model. In order to obtain a complete navigational model, analysts must manually refine and complete the obtained skeleton. To facilitate this, we introduce next several aspects that cannot be systematically identified by the traceability rules presented above:

1. **Temporal Relationship Semantics.** As we have seen during the presentation of the task taxonomy traceability rules some aspects of the temporal relationship semantics are not captured by the abstract primitives provided by the OOWS method.
2. **Function system actions as initial nodes.** Function system actions are derived into class operations by Rule S3a when they are activated from an

5. Requirements Traceability

Output IP. This rule cannot be applied when function system actions constitute the initial step of a task. In these situations, it is not possible to systematically detect the class in which the operation must be created. Then, operations that support these system actions must be manually defined by analysts. In this context, access to these operations must be also manually included in the navigational model.

3. **Subsystems.** Subsystems are defined when Rule N8 or Rule N10 are applied. The name that is assigned to these subsystems is systematically defined from the name of the source task. In this context, the defined name may not represent properly the content of the subsystem. Thus, analysts should manually modify these names in order to properly adjust them with the subsystem content.

Furthermore, as we have explained in the explanation of these rules the subsystem is created in order to isolate the navigational contexts that support two specific tasks from the contexts derived from the rest of tasks. However, if there are no navigational contexts derived from other tasks the subsystem may not be required. This analysis is not performed systematically by the traceability rules. Thus, analysts should manually analyze the navigational structure obtained after applying the whole set of traceability rules and remove the subsystems that they do not consider necessary.

4. **Double arrowed arcs between Output IPs.** This type of arcs is used to provide access to an Output IP from another Output IP. Furthermore, this type of arcs introduces an additional semantics: once users have accessed the second IP they must return to the first IP in order to finish the task. At the conceptual level, this *mandatory return* must be supported by a navigational link between two navigational contexts. However, this navigational link must carry out no information from one context to another. As we have explained above, this type of links are not supported by the OOWS navigational model primitives.
5. **Complementary classes connected to Complementary Classes.** Traceability rules allow us to detect complementary classes that are connected to the manager class. However, these rules are not able to detect complementary classes that are connected to other complementary classes. These complementary classes must be manually defined.
6. **Complementary class operations.** Traceability rules allow us to detect operations of the manager class. However, operations of complementary classes cannot be systematically detected. They must be manually defined by analysts.

5. Requirements Traceability

7. **Link attributes.** In order to properly define a navigational link we must indicate: (1) the target context that users access when the link is activated and (2) the element that constitutes the anchor in the source context and which allow users to activate the link. As we can see in Appendix B, OOWS allow us to define this anchor from an attribute of a navigational class. This attribute is the link attribute. Traceability rules presented above do not allow us to systematically detect link attributes. They must be defined by analysts manually.
8. **Operations that are related to the task logic.** We have seen in the derivation of the structural model that function system actions which are related to the task logic (instead of to an entity) cannot be systematically derived into class operations. In this context, if function system actions are not properly supported in the structural model we cannot derive systematically the navigational context view in which these operations are included. This aspect must be manually defined by analysts.
9. **Conditioned Flow.** The step flow defined in an elementary task performance is supported at the conceptual level by means of navigational links. However, the OOWS navigational model does not allow us to define navigational links with conditions.
10. **Type of User Roles and Change of User Role in Run Time.** From the type of users associated to the elementary tasks we can derive the user roles proposed by the OOWS method. These roles represent the profiles of the users that must use the Web application. The OOWS method allows us to define two kinds of user roles: Anonymous and Registered. This classification of user roles cannot be systematically derived from the requirements model. Analysts must define them manually.

Furthermore, if we analyze the task taxonomy in detail we can see how some tasks that are related by means of a temporal relationship must be performed by different type of users. For instance, the task Checkout must be performed by Customers. This task must be performed always after the task Collect Product which can be performed by Visitors or Customers. In this case, if the task Collect Product is performed by Visitors we must define a *Change of Role* that allows Visitors to change to Customers in order to perform the task Checkout. These mechanisms are not systematically derived by the traceability rules and they must be manually defined by analysts.

5.4 Conclusions

In this chapter, we have presented a set of traceability rules that allow analysts to systematically derive a skeleton of the OOWS conceptual schema from the requirements model. This skeleton constitutes a valuable starting point to perform the activity of conceptual modelling.

These traceability rules constitute a particular interpretation of the task-based requirements models. However, other interpretations are also acceptable. In fact, some OOWS conceptual primitives derived from these traceability rules are marked as proposed (with weak traceability) indicating that analysts can change them without critical consequences.

Finally, we want to remark that these traceability rules can be also considered from the point of view of model weaving. Model weaving is a recent research topic which proposes the use of models to define relationships between other models [Didonet Del Fabro et al. 2005]. These relationships can be used for different purposes: model composing, interoperability, data integration, ontology alignment or even traceability. In this context, the set of rules presented in this chapter can be considered to be a model weaving with traceability purposes between task-based requirements models and OOWS conceptual models.

A meta-model for weaving models is presented in [Didonet Del Fabro et al. 2006]. In this meta-model, the different type of relationships between models are characterized by means of the concept of link. A link represents a relationship between models. A weaving model is made up of links. Links are basically defined from a source model element and a target model element. The traceability rules presented in this chapter can be transformed into links by considering the field *TDE* to be the source model element and the field *MapsTo* to be the target model element.

Chapter 6

*“Some painters transform the sun into a yellow spot,
others transform a yellow spot into sun”*

Pablo Picasso
(Spanish Painter and Sculptor, 1881-1973)

6 Applying Traceability Rules through Model-to-Model Transformations

The previous chapter has introduced a set of traceability rules that allows us to analyze task-based requirements models in order to derive the main components of a Web application conceptual model. In the current chapter, we introduce a technique for automating this derivation. This technique is based on the definition of a set of model-to-model transformations (based on the traceability rules) and its further automatic application.

The technique that we propose to perform model-to-model transformations allows us to automatically obtain a skeleton of the OOWS conceptual model that satisfies the requirements captured in the task-based requirements model. Taking into account that the OOWS method is supported by a CASE tool that generates Web application prototypes from conceptual models (see Appendix B), the proposed model-to-model transformations are implicitly allowing us to automatically generate Web application prototypes from requirements models.

Section 6.1 introduces some background about model-to-model transformation techniques. This section also introduces the technique that we propose to use in this

6. Applying Traceability Rules through Model-to-Model Transformations

thesis. This technique is based on graph transformations and the use of the tool AGG [AGG]. We also explain the reasons of using this technique. Section 6.2 explains how the model-to-model transformations that support the traceability rules presented in Chapter 6 are defined by means of graph transformations. Section 6.3 explains how the model-to-model transformations are automatically applied. The conclusions of this chapter can be found in Section 6.4. Additionally, the underlying formalism in which the graph transformation technique relies can be found in Appendix C.

6.1 Model-to-Model Transformations

Model-to-Model Transformation has received a lot of attention in the recent literature related to Model Driven Development (MDD) [Gerber *et al.* 2002] [Sendall & Kozaczynski 2003] [Czarnecki & Helsen 2003]. MDD is the notion that we can construct a model of a system that we can then transform into a real thing [Mellor *et al.* 2003]. The software development's primary focus and products are models rather than computer programs.

Models are used to better manage a complex system description by using different models to capture the different aspects of the solution. However, we can use models not only horizontally to describe different system aspects but also vertically, to be refined from higher to lower levels of abstraction. At the lowest level, models use implementation technology concepts.

Thus, for the MDD vision to become reality, model transformations are a key factor. Within model transformations we can distinguish between model-to-code transformations and model-to-model transformations. Several techniques that allow us to perform both types of transformations have been proposed throughout the published literature. They have been surveyed in [Gerber *et al.* 2002] [Sendall & Kozaczynski 2003] [Czarnecki & Helsen 2003]. In this thesis, we focus on model-to-model transformations. The most relevant techniques for model-to-model transformations are the following:

- **Direct model manipulation:** These approaches usually offer an internal model representation together with mechanisms to manipulate this representation using a set of procedural APIs. They are usually implemented as an object-oriented framework, which may also provide some minimal infrastructure to organize the transformations (e.g., abstract classes for transformations)

One advantage of the direct-model manipulation approach is that the language used to access and manipulate the exposed APIs is commonly a general-purpose language such as Java, so the developers need little or no extra training to write transformations. Furthermore, developers are generally more comfortable with encoding complicated (transformation) algorithms in

6. Applying Traceability Rules through Model-to-Model Transformations

procedural languages. A disadvantage is that the APIs usually restrict the kind of transformations that can be performed. Also, because the programming languages are general-purposed, they lack suitable high-level abstractions for specifying transformations. Consequently, transformations can be hard to write, comprehend, and maintain.

Examples of this type of transformation are Jamda [Boocock 2003], UMLAUT [Ho *et al.* 1999], dMof [DMOF 2002] or Univers@lis [Universalis].

- **Relational approaches:** They are based on the definition of relations between element types of the target model and element types of the source model. These relations are defined by means of constraints which, in its pure form, are non-executable. However, if constraints are declaratively defined by using for instance logic programming (Prolog, F-Logic, etc) we can obtain execution semantic. In fact, logic programming is usually used to implement the relational approach because of characteristics such as unification-based matching, search or backtracking. In this case, predicates are used to describe the relations. Finally, remark that relational approaches require a clear separation of the source and target models.

Examples of this type of transformations are [Akehurst *et al.* 2003] and [Gerber *et al.* 2002].

- **XML-Based Approaches:** Since models can be serialized as XML documents using the XML Metadata Interchange (XMI) [XMI], it is possible to use existent XML tools, such as XSLT [XSLT] to perform model-to-model transformations.

XSLT is a language for transforming XML documents into other XML documents, which, in this case, represent models. A transformation in the XSLT language is expressed as a well-formed XML document. A XSLT transformation describes rules for transforming a source tree into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The result tree is separate from the source tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added.

The use of this kind of techniques to perform model-to-model transformations has several disadvantages [Gerber *et al.* 2002]. Even though XSLT was defined specially for describing transformations, it is nevertheless tightly coupled to the XML that it manipulates. Consequently, it requires experience and considerable effort to define even simple model transformations in XSLT. Furthermore, manual implementation of model-to-model transformations in

6. Applying Traceability Rules through Model-to-Model Transformations

XSLT quickly leads to non-maintainable implementations because of the verbosity and poor readability of XMI and XSLT.

- **Structure-Driven Approaches:** Approaches in this category define model-to-model transformations in two main steps: first, we must create the hierarchical structure of the target model; second, we must set the attributes and references in the target model from the information that is defined in the source model.

Generally, approaches of this type are supported by a framework that provides us with constructors to support both steps. The framework is also who control the transformation scheduling. An example of this type of approach is the transformation framework of OptimalJ [OptimalJ]. This framework provides users with classes from which they must create subclasses in order to create transformation rules. A transformation rule is implemented by a method whose input parameter constitutes an element of the source type and the return constitutes an element of the target model.

These approaches were mainly developed in the context of Enterprise JavaBeans (EJB) [EJB] and databases schema generation from UML models. They strongly support 1-to-1 and 1-to-n correspondences between source and target. It is unclear how well these approaches can support other kinds of applications.

- **Hybrid Approaches:** Hybrid approaches combine characteristics from the previous approaches. In this category, we can find transformations languages such as the Transformation Rule Language (TRL) [TRL 2003] or Atlas Transformation Language (ATL) [Bézivin et al. 2003] which can be considered as a combination of imperative and declarative approaches. In TRL we can define mappings that constitute relationships between elements of the source and target models that are constrained by a set of invariants (similar to the relations in the relational approach). However, operational rules can be also defined. These rules represent executable transformation rules that explicitly states whether a rule creates, updates or deletes elements. The scheduling is explicitly defined by defining in the body of a rule calls to other rules. ATL is very similar to TRL. ATL allows us to define declarative, hybrid and imperative rules. For instance, hybrid rules are defined from declarative patterns that match with the source and target models and these patterns are complemented with imperative logic.

In the same way, Query/View/Transformation (QVT) Specification for MOF proposed by the OMG [QVT] also present a hybrid declarative/imperative nature. The declarative part is split into a user-friendly part based on transformations which comprises a rich graphical and textual notation, and a core part which provides a more verbose and formal definition of the

6. Applying Traceability Rules through Model-to-Model Transformations

transformations. The declarative notation is used to define the transformations that indicate the relationships between the source and target models, but without specifying how a transformation is actually executed. QVT also defines operational mappings that extend the meta-model of the declarative approach with additional concepts. This allows for a definition of transformations that use a complete imperative approach.

- **Graph Transformations:** Graph transformations have been used for many years to represent transformation systems. The first step in this type of transformation consists in represents the source and the target elements of the transformation (models in our case) as graphs. Then, a set of graph transformation rules are applied to the source graph in order to obtain the target graph. To do this, each graph transformation rule is defined as a rewriting rule that selects a sub-graph of the source model and applies to this sub-graph any type of transformation (adding, deleting or modifying a node or an edge). Graph transformations have been used in many fields of software engineering for: software refactoring [Mens et al. 2001], software evolution [Heckel et al. 2002], multi-agent system modeling [Depke et al. 2002], modeling language formalization [Varró 2002] or user interfaces development [Limbourg 2004].

6.1.1 Graph Transformations as Model-to-Model Transformation Technique

In this thesis, we use a technique based on graph transformations in order to perform the model-to-model transformation between task-based requirements models and OOWS conceptual models.

The main reasons for this choice are that graph transformations are:

- **Formal:** Graph transformations are based on a sound mathematical formalism (graph theory) and enables verifying formal properties on represented artifacts.
- **Visual:** Graph-transformation approaches are capable of expressing model transformation in a declarative manner but using a graphical syntax which facilitates the comprehension of its internal logic and its application.
- **Correctness checkable:** Several mechanisms (such as conditional graph rewriting or typed graph rewriting) are provided in order to check the correctness of the artifact obtained after a transformation.
- **Modular:** Graph transformations allow us to fragment complex transformation into a set of smaller ones. The fact that graph transformations have no-side effects facilitates this fragmentation.

6. Applying Traceability Rules through Model-to-Model Transformations

- **Tool Supported:** Several tools such as AGG [AGG], AToM3 [De Lara & Vangheluwe 2002], VIATRA [Varró & Pataricza 2004] or VMTS [VMTS] can be used for defining and applying graph transformation. These tools have been tested throughout multiple applications in real scenarios. This was not the case of the other approaches when our work was initiated.

6.2 From Requirements Models to Conceptual Models throughout Graph Transformations

In this section, we present a practical use of graph transformation notions in order to automatically obtain OOWS conceptual models from task-based requirements models. First we introduce the main aspects of the graph transformation theory. Next, we show how traceability rules presented in the previous chapter are implemented as graph transformations.

6.2.1 Graph Transformations in a Nutshell

Graph transformations rely on the theory of graph grammars [Rozenberg 1997]. A graph grammar is defined from: (1) a transformation system, which is a set of transformation rules and (2) a graph to which the transformation rules are applied (called host graph).

A rule is a graph rewriting rule that is made up of a Left Hand Side (LHS), which represent a sub-graph of the host graph, and a Right Hand Side (RHS), which represent a sub-graph of the target graph. Additionally, Negative Application Condition (NAC), which represent a sub-graph that must not be contained in the host graph and Positive Application Conditions (PAC), which represent additional conditions that must be satisfied, can be also defined. Figure 6.1 illustrates how a rule of a transformation system is applied to a graph G : when the LHS matches G , and the NAC does not match G , and the PAC is accomplished, then the LHS is replaced by the RHS. G is transformed into G' . All elements of G not covered by the match are considered as unchanged. All elements contained in the LHS and not contained in the RHS are considered as deleted.

To add more expressiveness to transformation rules, variables may be associated to attributes within a LHS. These variables are initialized in the LHS and their value can be used to assign an attribute to the expression of the RHS. An expression may also be defined to compare a variable declared in the LHS with a constant or with another variable. This mechanism is called ‘attribute condition’ [Partsch & Steinbruggen 1983]. It can be used to define conditions in PACs.

6. Applying Traceability Rules through Model-to-Model Transformations

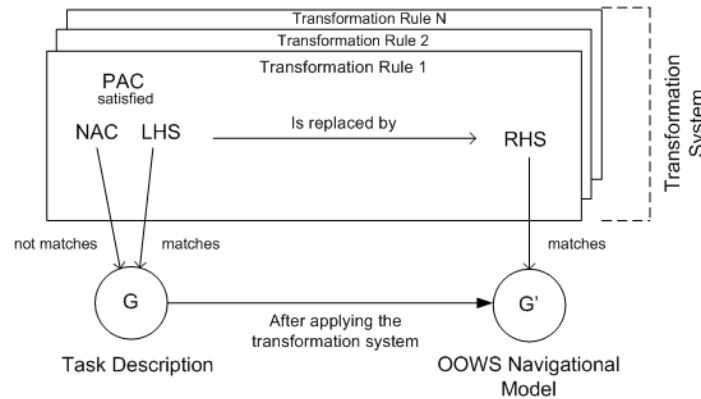


Figure 6.1 A transformation system

Furthermore, the host and the target graphs as well as the graph transformation rules must be defined according to a predefined graph structure. This graph structure is defined as a *type graph*. A type graph is a graph that indicates the nodes and edges that can be used to construct graphs and graph transformation rules (from a model perspective, this type graph represents a meta-model). The part of the type graph that indicates how the host graph must be defined can be also used to define pre-conditions (such as cardinality constraints) that must be satisfied before performing the transformation process. The part of the type graph that indicates how the target graph must be defined can be also used to define post-conditions that must be satisfied after performing the transformation process.

For more information about the theoretical notions of graph transformations see Appendix C.

6.2.2 Defining Graph Transformations

We define the traceability rules presented in Chapter 6 as graph transformations. These traceability rules take as source a task-based requirements model in order to derive a skeleton of two models of the OOWS conceptual schema: (1) the structural model and (2) the navigational model.

In this context, we can gather the set of defined graph transformations in two main groups depending on the OOWS model that they derive. Each of these groups are used to define a graph grammar. Each of these graph grammar can be considered to be a model-to-model transformation. Thus, we have two model-to-model transformations:

- A model-to-model transformation between the task-based requirements model and the OOWS structural model.

6. Applying Traceability Rules through Model-to-Model Transformations

- A model-to-model transformation between the task-based requirements model and the OOWS navigational model.

Thus, in order to define a model-to-model transformation we must define a graph grammar. To do this, the next steps must be followed:

- First, we must define the type graph (meta-model) according to which the host and the target graph as well as the graph transformations rules must be defined.
- Next, the source and target models must be represented as graphs.
- Finally, we must define the set of graph transformation rules that must be applied to the source graph in order to transform it into the target graph.

Next, we explain how type graphs, graphs and graph transformation rules are defined in order to create the two model-to-model transformation introduced above. We use the AGG environment [AGG], then we use the visual syntax proposed by this tool.

Along the next subsections we partially present the definition of the model-to-model transformation between the task-based requirements model and the OOWS navigational model. The whole definition of this transformation as well as the whole transformation between the task-based requirements model and the OOWS structural model have been omitted in order to not overload this chapter. They can be found in [Valderas 2007b].

Defining Type Graphs

A type graph is defined by means of an *attributed type graph* (see Appendix C). The basic idea of type graphs is to define a graph whose vertices represent types for the vertices of other graphs and whose edges represent types for edges of other graphs. An attributed graph is a graph in which we attach attributes to its vertices and edges. Each vertex and edge can have attached more than one attribute. An attributed type graph is a type graph in whose types we can attach attributes.

Figure 6.2 shows an example of attributed type graph. In these graphs, vertices are depicted as boxes. The upper side of the box specifies a label that defines the name of the vertex type. The lower side of the box contains the definition of the attributes that the vertex type has. They are defined as a couple *attribute type - attribute name*. Furthermore, vertex multiplicity constraints are indicated in right upper side of the box. These constraints allow us to indicate how many vertices of a specific type can be created.

Edges are represented as directional arrows. They present a label which indicates the type of the edge. Its associated attributes are depicted below this label. Finally, edge

6. Applying Traceability Rules through Model-to-Model Transformations

multiplicity constraints are defined by using the notation proposed for UML class diagrams [UML]. These constraints allow us to indicate how many vertices may be connected through an instance of an edge type.

Furthermore, notice that we use inheritance of vertex types. In order to graphically indicate this inheritance, a UML generalization relationship is used. If an inheritance relationship is defined from a vertex A to a vertex B, the vertex B inherits all the attributes defined for the vertex A as well as the edges that are connected to the vertex A. Additionally, types can be declared as abstract ones. Abstract types are those used only for inheritance purposes. They cannot be used to create vertices in the source or target graphs. Abstract types are graphically depicted with italic font and by putting their associated label between brackets. Abstract types are graphically depicted with italic font and by putting their associated label between brackets.

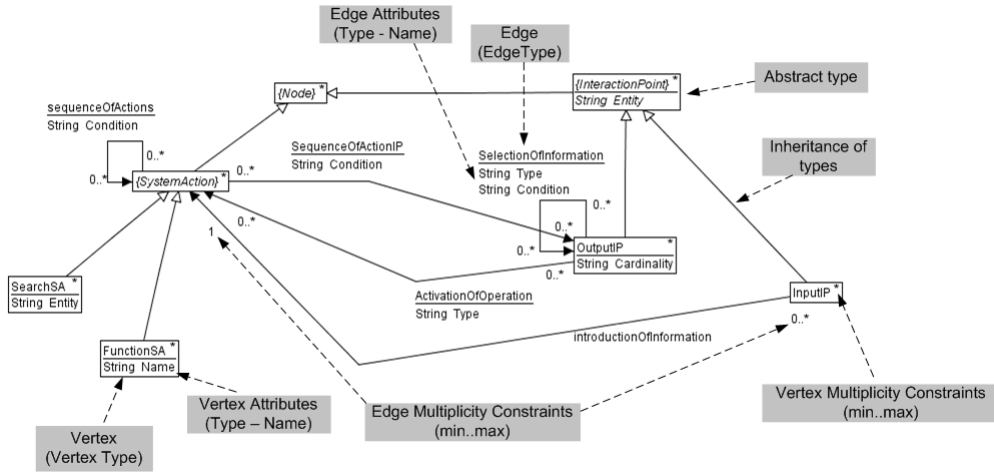


Figure 6.2 Example of a type graph

Figure 6.2 shows a partial view of the type graph for representing task-based requirements model. A more detailed description of this type graph is introduced in Figure 6.4.

The type graph that we use in each model-to-model transformation (hereafter know as TG) constitutes the type graph of both the source graph and the target graph. The type graph of the source graph is specified by a subset of elements T_S specified in the type graph TG. Correspondingly, the type graph of the target graph is specified by another subset of elements T_T specified in the type graph TG. From a model perspective, T_S constitutes the meta-model of the source model and T_T constitutes the meta-model of the target model. However, the type graph TG not only includes T_S and T_T , but also additional types and relations which are needed for the transformation process only. These additional elements are used with two purposes: (1) Facilitate the management of traceability aspects and (2) support auxiliary graph structures that are needed to perform intermediate steps in the transformation process.

6. Applying Traceability Rules through Model-to-Model Transformations

The graph G_S that is taken as source in the transformation is typed over T_S (G_S is defined by using the types defined in T_S) but also over TG . In the same way, the graph G_T obtained after the transformation is typed over T_T and over TG . The different sub-graphs that are obtained from intermediate steps of the transformation are typed only over TG . This aspect is shown in Figure 6.3.

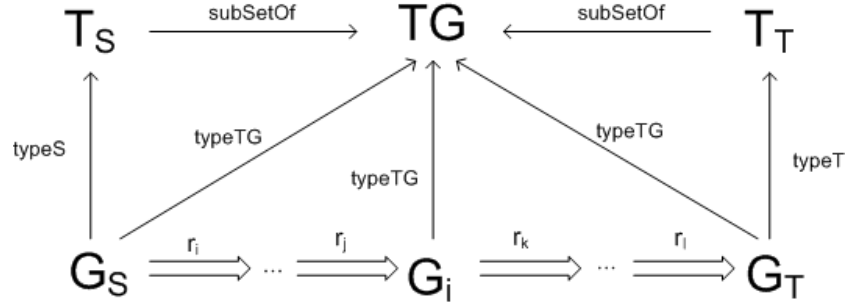


Figure 6.3 Relation between type graphs and graphs

Post-conditions (see Appendix C) are defined throughout the type graph. After each application of a rule the transformation engine check that the obtained graph is compliant with the pre-defined type graph. Thus, we can sure that the different graphs obtained during the whole graph transformation process are always syntactically correct. The correctness of the source graph is checked over the type graph defined by the subset T_S ; the correctness of the resultant graph is checked over the type graph defined by the subset T_T ; finally, the correctness of the different sub-graphs that are obtained from intermediate steps of the transformation process are checked over TG .

Figure 6.4 shows the type graph that supports the mode-to-model transformation between the task-based requirements model and the OOWS navigational model. The different elements of the type graph introduced in Figure 6.3 are defined as follow:

- The subset T_S (type graph for the source graph) is defined from the meta-model of the task-based requirements model. This meta-model is introduced in Appendix A. This sub-graph is shown in the upper side of Figure 6.4.
- The subset T_T (type graph for the target graph) is defined from the meta-model of the OOWS navigational model. This meta-model is introduced in Appendix B. This sub-graph is shown in the lower side of Figure 6.4.
- Additional types and relationship are defined to connect the elements of T_S to the elements of T_T . These additional elements are used to identify the application of a rule. Furthermore, in order to facilitate traceability aspects they are connected to the elements to which the rule is applied (by means of a *source* edge) and also to the elements created by the rule (by means of a *target*

6. Applying Traceability Rules through Model-to-Model Transformations

edge). In Figure 6.14 we only show the element S2F. The rest of elements are omitted in order to not overload this figure. They can be found in [Valderas 2007b]. The element S2F is created when the graph transformation rule that support the traceability rule N4c is applied. This rule derives search system actions (S) into (2) search filters (F).

Classes and relationships defined in the meta-models of both source and target models are represented in the TS and TT as follows:

Meta-Model Classes:

- Each class in a meta-model that represent a model element (hereafter known as *element classes*) is defined as a type vertex. See in TS type vertices such as Elementary Task, Output IP or Search SA that represents elements of the task-based requirements model. In the same way, in TT we can find type vertices such as NavigationalContext, NavigationalClass or Index that represent elements of the OOWS Navigational Model. Attributes of these classes are defined as attributes of the corresponding type vertices.
- Each class in a meta-model that represent a relationship between model elements (hereafter known as *relationship classes*) is defined as an edge. See in TS edges such as TaskRefinement, SelectionOfInformation or SequenceOfActions that represent classes in the task-based requirements meta-model that are defined for capturing relationship between model elements. In the same way TS presents edges such as ServiceLink that represents a class in the OOWS navigational meta-model that defines a relationship between navigational contexts and navigational operations. Attributes defined in these classes are defined as attributes of the corresponding edges.

6. Applying Traceability Rules through Model-to-Model Transformations

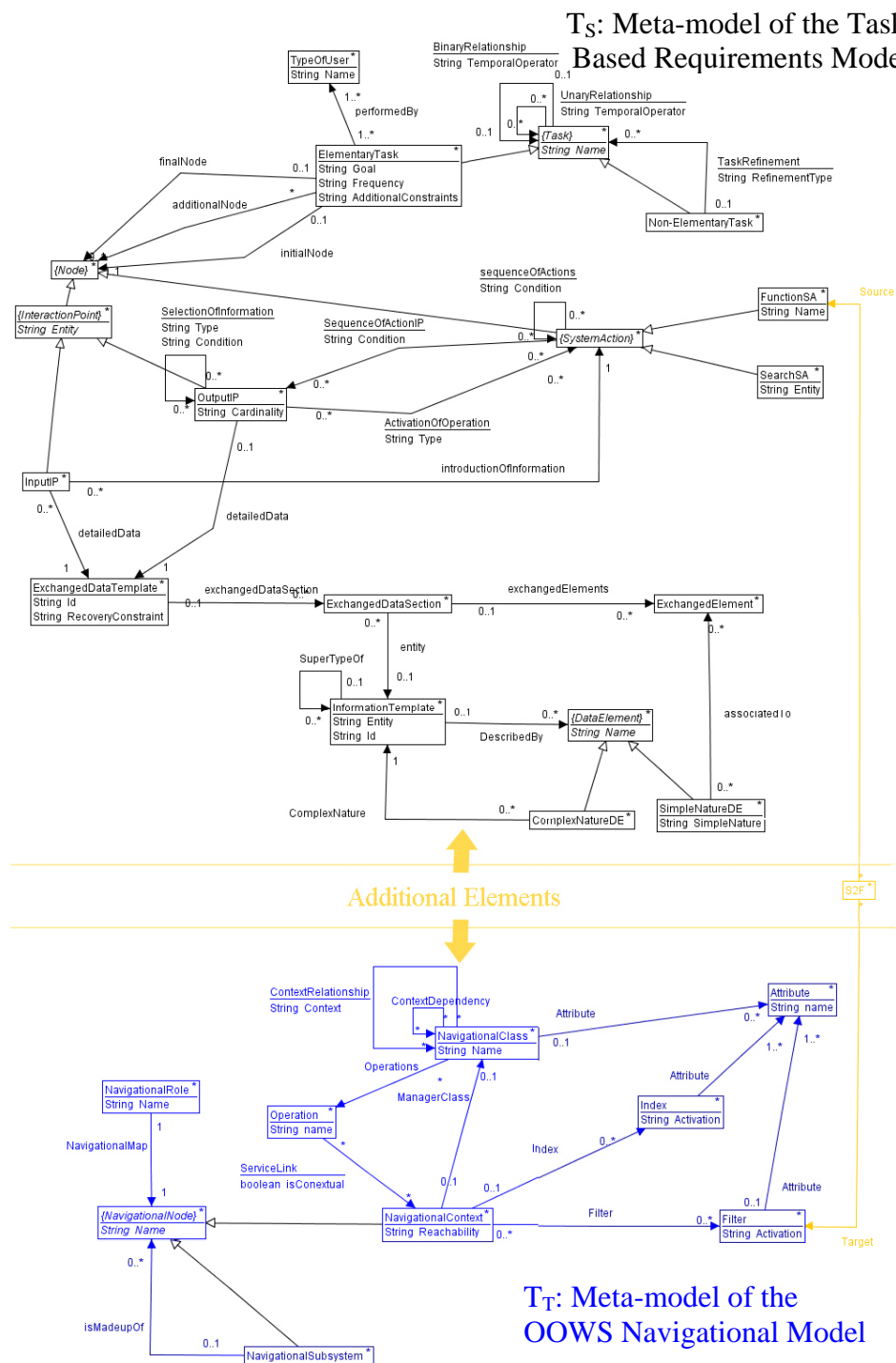


Figure 6.4 The type graph used in this thesis

6. Applying Traceability Rules through Model-to-Model Transformations

Meta-Model Relationships:

- Aggregation and association relationships defined between element classes are represented in the type graph by means of edges between the corresponding type vertices. See in TS edges such as `ExchangedDataSection`, `ComplexNature` or `SuperTypeOf`. These edges represent the relationships between the `ExchangedDataTemplate` and `ExchangedDataSection` classes, the `ComplexNatureDE` and `InformationTemplate` classes, and a reflexive relationship defined in the class `InformationTemplate`, respectively. In the same way, we can see in TT edges such as `ExplorationLink`, `ManagerClass` or `Operations` that represent the relationships between the classes `NavigationalRole` and `NavigationalContext`, `NavigationalContext` and `NavigationalClass`, and `NavigationalClass` and `NavigationalOperation`, respectively.
- Generalization relationships defined between element classes are kept in the type graph by means of inheritance relationships between the corresponding type vertices. In the same way, abstract concepts defined in a meta-model are represented in the type graph as abstract type vertices. See in TS the abstract vertex `Task` that represents the abstract element class `Task`. We can also see how the generalization relationship defined from this class is also supported in the type graph (the type vertices `ElementaryTask` and `Non-ElementaryTask` inherit the attributes and edges defined for the abstract type vertex `Task`). In the same way, we can see in TT the abstract type `NavigationalNode` that represents the corresponding abstract class in the meta-model of the OOWS navigational model. The generalization defined from this abstract class is supported in the type graph by means of an inheritance relationship defined between the abstract type vertex `NavigationalNode` and the type vertices `NavigationalContext` and `NavigationalSubsystem`.
- Generalization relationships defined between relationship classes are not supported in the type graph. As we have explained above, these classes are represented by edges. Inheritance relationships among edges cannot be defined in a type graph. Thus, we define an edge for each child class defined in the generalization relationship. These edges present the attributes of the class that they represent plus the attributes of the parent class. For instance, in the meta-model of the task-based requirements model we can find the relationship class `SequenceOfNode` (which present an attribute *condition*) that is specialized in the relationship classes `SequenceOfActionIP` and `SequenceofActions`. In TS we can find the edges `SequenceOfActionIP` and `SequenceofActions` that represent the corresponding relationship classes. However, and edge for representing the relationship class `SequenceOfNode` is not defined. Notice moreover how the edges `SequenceOfActionIP` and `SequenceofActions` present both the attribute *condition*.

6. Applying Traceability Rules through Model-to-Model Transformations

Defining Graphs

In order to define graphs we use *attributed*, *typed*, *identified* and *direct graphs* (see definition 8 in Appendix C). Figure 6.5 shows a partial view of a graph that represents a task-based requirements model. The visual syntax used for defining these graphs is the following: Vertices are depicted as boxes. Their type is indicated in the upper side of boxes. Attributes of a vertex are depicted in the lower side of the box that represents the vertex. Each attribute is a couple of *attribute name* - *attribute value*. Edges are represented as directional arrows. Their type is represented as their label. Attributes of edges are depicted below their types.

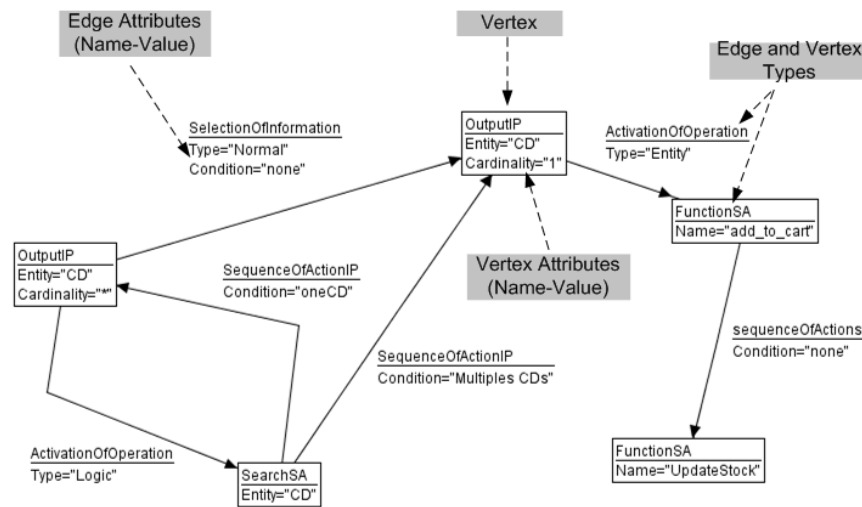


Figure 6.5 Partial view of a task based requirements model represented as a graph

Graph in Figure 6.5 represents part of the performance description associated to the elementary task Add CD (see Figure 4.20). In particular, this graph represent the IP Output (CD,*) from which users can either activate a search system action or access the IP Output(CD,1). In this IP users can activate the system action add_to_cart. After this operation the system performs the action UpdateStock.

Defining Graph Transformation Rules

As we have explained above, graph transformation rules are defined by means of two sub-graphs: a LHS sub-graph and a RHS sub-graph. We complement rules with Negative Application Conditions (NACs) or Positive Application Conditions (PACs). NACs are defined by means of another sub-graph. PACs are defined by expressions that compare variables with constants or with other variables.

6. Applying Traceability Rules through Model-to-Model Transformations

The visual syntax used to define each sub-graph of a graph transformation rule (LHS, RHS and NACs) is the same as the one presented for graphs. Only a difference is introduced: in order to graphically indicate the mapping of a graph element of the LHS with a graph element of the RHS or the NAC, numerical labels are used. Next, we explain this aspect in detail as well as other ones related to the definition of graph transformation rules. To do this, we present some examples of the graph transformation rules that implement the model-to-model transformation between task-based requirements models and OOWS navigational models. The set of graph transformations that completely implements this model transformation can be found in [Valderas 2007b].

Basic Rules. Figure 6.6 shows an example of a basic transformation rule that is defined with a LHS and a RHS. This graph transformation rule supports the traceability rule N1a (see Chapter 5). This rule derives navigational roles (of the OOWS user diagram, see Appendix B) from the types of user associated to the different elementary tasks. The LHS of this rule matches with an `ElementaryTask` and a `TypeOfUser` that is associated to it. The RHS creates a `NavigationalRole` and connect it to the `TypeOfUser` from which it is derived. To do this, the additional element `TU2NR` is created.

Notice how graph elements are marked with numbers. A graph element that is marked with the same number in both parts of the rule (LHS and RHS) is preserved (vertices `ElementaryTask` and `TypeOfUser`, and the edge `performedBy`). If a graph element of the LHS is not marked, it is deleted. If a graph element of the RHS is not marked, it is an element that has been newly created (vertices `NavigationalRole` and `TU2NR`, and edges `source` and `target`).

Finally, we use a variable in this rule (*username*). This variable is initialized in the LHS (it takes the name of the type of user) and it is used in the RHS to assign a name to the newly created navigational role.

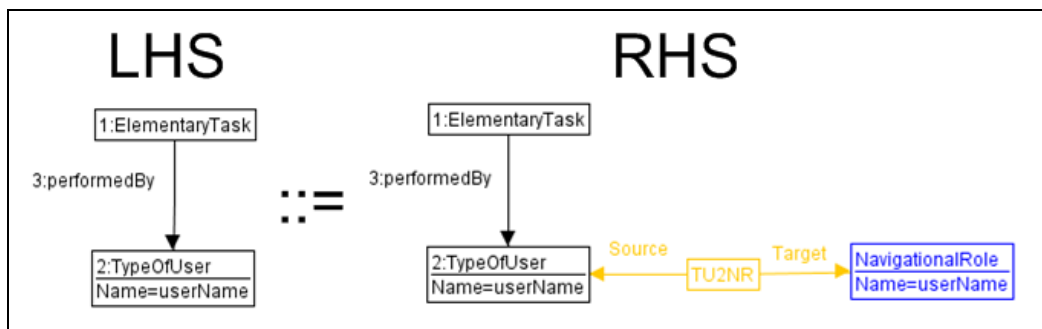


Figure 6.6 Graph transformation rule for traceability rule N1a

6. Applying Traceability Rules through Model-to-Model Transformations

Rules with NACS. Another aspect to be considered in the creation of graph transformation rules is the definition of NACs. We define NACs in the same way that we define the LHS and RHS parts of a rule. For instance, Figure 6.7 shows the graph transformation rule that supports the traceability rule N2 (see Chapter 5). This rule derives both a navigational context and a navigational class from an Output IP, except from graph structures that are derived into indexes.

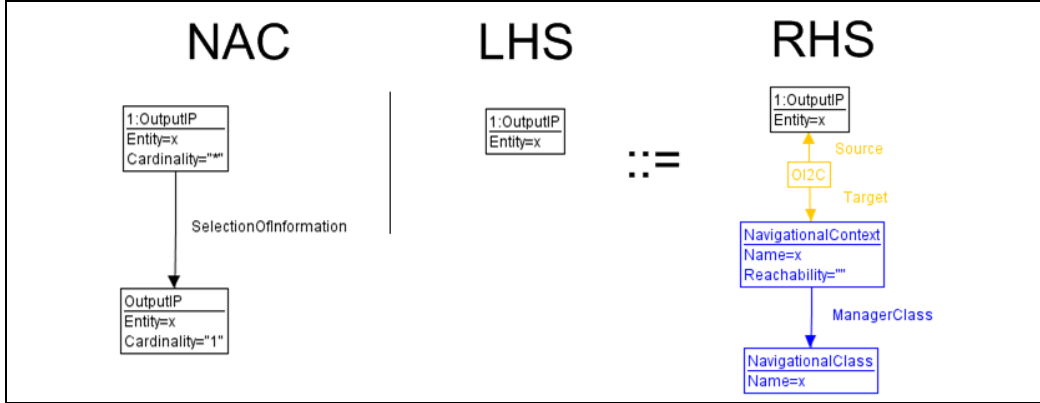


Figure 6.7 Graph transformation rule for traceability rule N2

The LHS of this rule matches with an OutputIP. The RHS creates a NavigationalContext and a NavigationalClass. The NavigationalContext is connected to the NavigationalClasses throughout the edge ManagerClass. Furthermore, the RHS also creates the additional graph element OI2C, which associates the Output IP to the navigational context that is derived from it. The variable x is used to obtain the entity of the OutputIP (in the LHS) and then using it to name the NavigationalContext and the NavigationalClass (in the RHS).

A NAC is defined in this rule to check that the OutputIP that match with the LHS is not connected to another OutputIP with the same entity and cardinality 1. Notice how to indicate that the Output IP defined in the NAC is the same as the one defined in the LHS they are marked with the same number (the number 1). The cardinality is checked by assigning a value to the corresponding vertex attribute (value “*” or value “1”). As far as the entity, it is checked by assigning the value of the variable x (initialized in the LHS) to the attribute Entity of the two OutputIPs defined in the NAC (this checks that both IPs have the same entity).

Rules with PACS. We also can use variables in order to define positive application conditions. We can compare a variable declared in the LHS with a constant or with another variable. This mechanism is called “attribute condition”. An example of this is shown in Figure 6.8. This figure shows the graph transformation rule that supports the traceability rule N4a. This rule derives an index from those graph structures that represent two connected Output IPs. These two IPs must provide information about

6. Applying Traceability Rules through Model-to-Model Transformations

the same entity but the first one must have a cardinality multiple and the second one must have a cardinality of one.

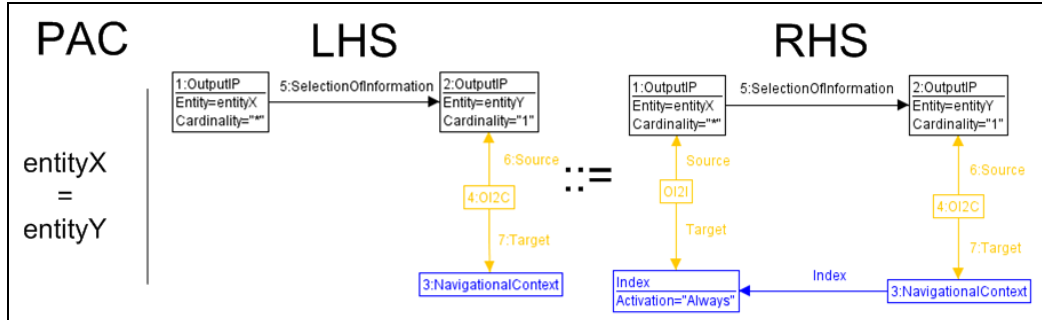


Figure 6.8. Graph transformation rule for traceability rule N4a

The LHS of this rule matches with two connected OutputIP. The additional element OI2C is used in the LHS in order to identify the navigational context that has been derived from the second Output IP when the Rule N2 is applied. In order to indicate the cardinality of both OutputIPs we assign the corresponding value (* in the first OutputIP, and 1 in the second one) to the *Cardinality* attribute. In order to indicate that both IP must be associated to the same entity we use a PAC that is defined from the variables *entityX* and *entityY*. These variables are initialized in the LHS. Then, we indicate in the PAC that both variables must have the same value. Notice how variables cannot be used in the LHS to indicate match properties (as we have done in the NAC of the rule presented above). Variables are used in the LHS only to be initialized.

The RHS of this rule creates an Index. The activation of the Index is “Always”. The Index is connected to the NavigationalContext throughout the edge Index. Furthermore, the RHS also creates the additional graph element OI2I that associates the first Output IP to the index that is derived from it.

Rules with Abstract Types. We have seen in the previous section that we can define abstract types in the type graph. These types are used for inheritance purposes and they cannot be used for creating vertices in a host graph. However, they can be used in the definition of graph transformation rules.

If we create a LHS with an abstract vertex, it will match with every element that is a child of the abstract vertex. See for instance the rule shown in Figure 6.9. This rule supports the traceability Rule N11. It defines navigational contexts as Exploration navigational contexts from the *Iterative Task* unary temporal relationship. However, Iterative Task relationships can be defined in both type of tasks, elementary tasks and non-elementary tasks. Thus, the rule presented in Figure 6.9 is defined by means of the abstract vertex Task. Then, this rule can be applied to both types of task.

6. Applying Traceability Rules through Model-to-Model Transformations

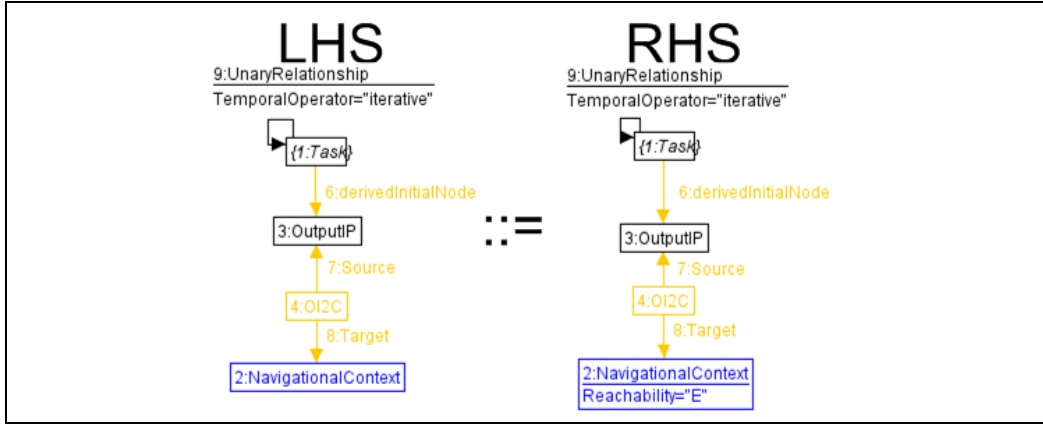


Figure 6.9 Graph transformation rule for traceability rule N11

The LHS of this rule matches with elementary tasks and non-elementary tasks. In this LHS, we have used two additional elements: (1) the auxiliary edge *derivedInitialNode* that is defined in order to indicate the OutputIP that constitutes the initial step of the task. The rule that creates this auxiliary edge can be found in [Valderas 2007b]; (2) the auxiliary vertex OI2C in order to identify the navigational context derived from the Output IP by applying Rule N2.

The RHS of this rule assigns a value to the Reachability attribute of the NavigationalContext. The value assigned is “E” (Exploration). Notice that this rule is also an example of how rules can be defined only to change attributed values, without changing the LHS graph structure.

Further Rules. In order to better illustrate how traceability rules are implemented by means of graph transformations we introduce next some other examples of graph transformation rules.

Figure 6.10 shows the graph transformation rule that supports the traceability rule N4b. This traceability rule derives index attributes from the entity features associated to the Output IP from which the index is derived.

The LHS of the graph transformation rule matches with a graph structure that allows us to know the entity features that are associated to an Output IP. The auxiliary element OI2I is defined in the LHS in order to only match with those Output IPs that have been derived into an index. A PAC is defined moreover to check that the feature and the Output IP are associated to the same entity (otherwise, another graph transformation rule is applied; it is shown in [Valderas 2007b]). The RHS creates the corresponding index attribute which is connected to the index. The auxiliary graph element ES2AI is created to associate the newly created index attribute to the SimpleNatureDE from which it is derived. The variable *att* is initialized in the LHS

6. Applying Traceability Rules through Model-to-Model Transformations

with the name of the simple nature feature. This name is used in the RHS to name the index attribute.

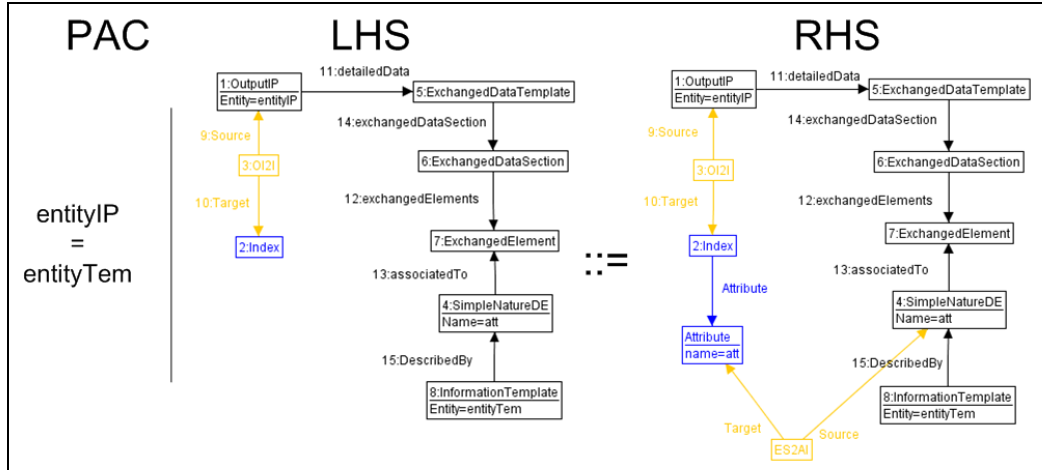


Figure 6.10 Graph transformation rule for traceability rule N4b

Figure 6.11 shows the graph transformation rule that supports the traceability rule N4c. This traceability rule derives search filters from those search system actions that are activated from an Output IP that has been derived into a navigational context. The activation mode of the derived search filters is “Never”.

The LHS of the graph transformation rule matches with a graph structure that represents an Output IP connected to a Search system action. The auxiliary element OI2C is included in order to match with those Output IPs that have been derived into a navigational context. The RHS creates a filter, and connect it to the navigational context derived from the Output IP. The auxiliary graph element S2F is created to associate the newly created filter to the search system action from which it is derived.

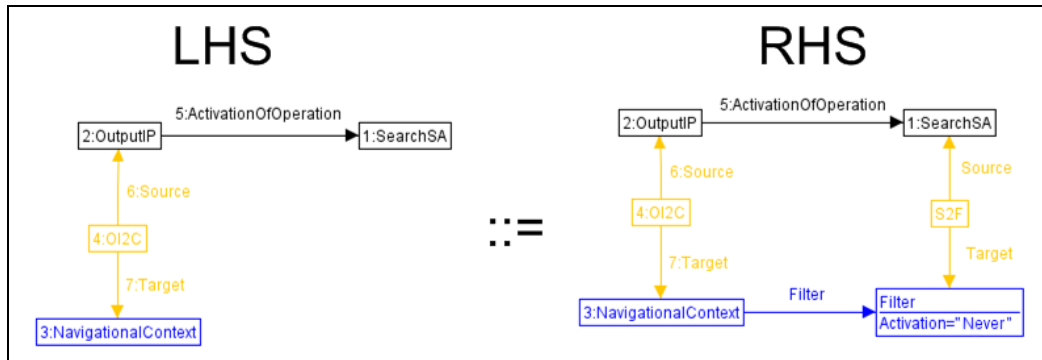


Figure 6.11 Graph transformation rule for traceability rule N4c

6. Applying Traceability Rules through Model-to-Model Transformations

Figure 6.12 shows the graph transformation rule that supports the traceability rule N5a. This traceability rule derives navigational context relationships (and complementary navigational classes) from those arcs that connect two Output IPs that have been derived into two navigational contexts. Each navigational relationship is defined in the navigational context derived from the Output IP connected to the arc's source. These navigational context relationships are implicitly defining navigational links between both navigational contexts.

The LHS of the graph transformation rule matches with a graph structure that represents two connected Output IPs. The auxiliary element OI2C is included in order to identify the navigational contexts (with their manager class) that have been derived from both Output IPs. The RHS creates a navigational class and connect it to the manager class of the first navigational context by means of a context relationship. The auxiliary graph element A2NL is created to connect the newly created navigational class to the two Output IPs from which it is derived. Furthermore, we use two variables: (1) *nameClass*, which is used to name the navigational class that is created, and (2) *nameContext*, which is used to define the value of the attribute *Context* in the navigational context relationship that is created.

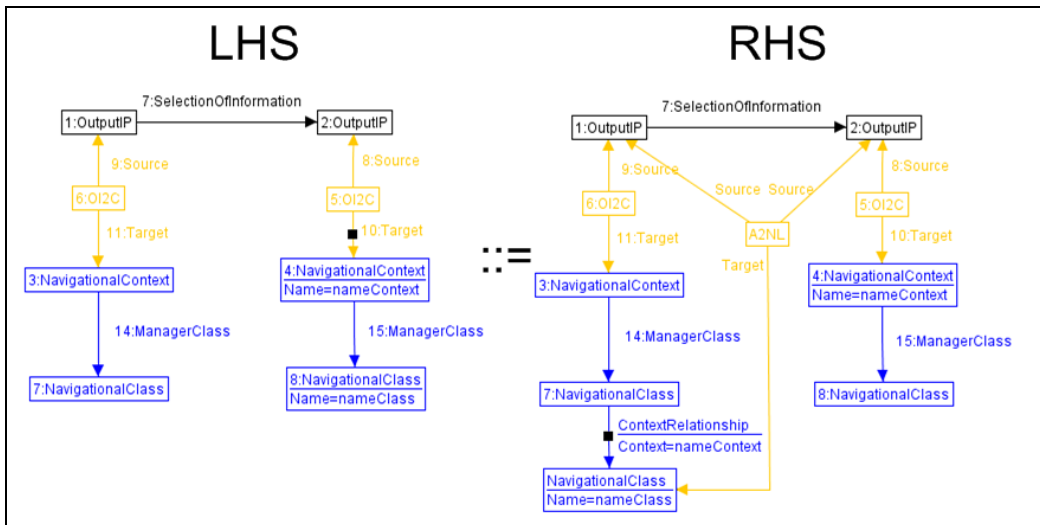


Figure 6.12 Graph transformation rule for traceability rule N5a

6.2.2 A Tool for Defining Graph Grammars

In order to define type graphs, graphs and graph transformation rules we use the AGG environment [AGG].

AGG (Attributed Graph Grammar system) can be considered to be a genuine programming environment based on graph transformations. It provides us with:

6. Applying Traceability Rules through Model-to-Model Transformations

- (1) A visual editor for defining type graphs, graphs and transformation rules.
- (2) An engine for automatically applying graph transformation rules. AGG present two modes: (1) *debug* mode, which allows us to apply each graph transformation rule individually, and (2) *interpretation* mode, which allows us to apply the whole sequence of rules.
- (3) Consistency checking. AGG checks the consistent of graphs and graph transformation rule in both transformation time and definition time. The consistence is checked upon a pre-defined type graph. Furthermore, AGG also allows us to define additional constraints by means of graph structures and first order logic.
- (4) Application of positive and negative conditions. AGG allow us to define NACs by means of graph structures. PACs can be defined by means of Java expressions.

Next, we describe the AGG user interface in detail as well as the strategy that this tool uses for storing graphs.

AGG User Interface

Figure 6.13 shows a snapshot of the AGG user interface. Frame 1 is the grammar explorer. It is possible to have more than one graph grammar loaded. The grammars and their contents are visualized by a tree where the graph grammar, the host graph or the rule that is selected is highlighted. The selected host graph or rule is shown in the corresponding graphical editor: In the upper right side of this figure frames 2, 3 and 4 allow us to define the elements of a graph transformation rule: a negative application condition (frame 2), a left hand side (frame 3) and a right hand side (frame 4). The host graph on which graph transformation rules will be applied is shown in Frame 5. Frame 5 is also used to define the type graph. The attribution of graph elements is done in a special attribute editor. It is shown in Figure 6.14.

The attribute editor in Figure 6.14 pops up when a graph element (either a vertex or an edge) is selected for attribution. The upper side of this figure shows the window that allows us to assign a value to an attribute, if the graph element that we are editing belongs to the host graph. If the graph element that we are editing belongs to the sub-graph that constitutes the LHS of a graph transformation rule, this window also allows us to assign a variable to an attribute. This variable can be used to define positive application conditions. To do this, we use the window that is shown in the lower side of the figure.

In the example of Figure 6.14, we are editing the attributes of an Output IP (Entity and Cardinality). This IP is used to define the LHS of a graph transformation rule. Notice how we assign the value *CD* (between colons) to the attribute Entity.

6. Applying Traceability Rules through Model-to-Model Transformations

However, we assign the variable *card* to the attribute Cardinality. Furthermore, we have defined a positive application condition based on this variable. The rule in which this Output IP is defined will be only applied if the cardinality of the Output IP is equals to asterisk (see the *Conditions* section of the window that is shown in the lower side of the figure: *card.equals("*")*).

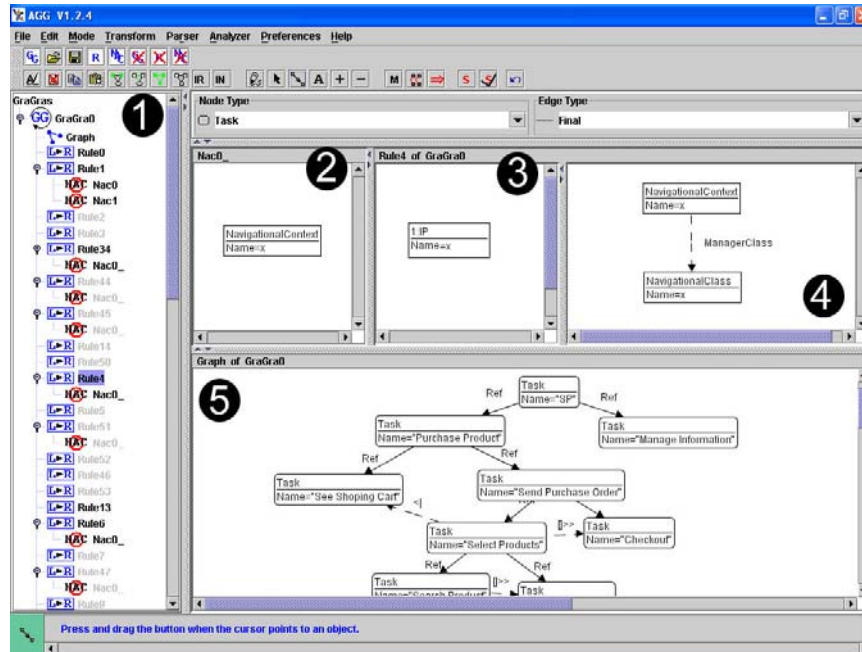


Figure 6.13 AGG user interface

Attribute Context		Current Attribute		Customize		
Shown	Handler	Type	Name	Expression	OK	
<input checked="" type="checkbox"/>	Java Expr	String	Entity	"CD"	<input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/>	Java Expr	String	Cardinality	card	<input checked="" type="checkbox"/>	
<input type="checkbox"/>					<input type="checkbox"/>	

Attribute Context		Current Attribute		Customize		
Parameters and Variables						
In	Out	Handler	Type	Name	Expression	OK
<input type="checkbox"/>	<input type="checkbox"/>	Java Expr	String	card		<input checked="" type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>					<input type="checkbox"/>

Conditions		
Expression	OK	
card.equals("*")	<input checked="" type="checkbox"/>	
	<input type="checkbox"/>	

Figure 6.14 AGG user interface for editing attributes

6. Applying Traceability Rules through Model-to-Model Transformations

Persistence of Graphs

Next, we explain how the AGG tool stores graphs. We explain this in order to better understand the proposed strategy for automating the application of model-to-model transformations that is presented in the next sub-section.

AGG uses XML-based documents in order to store graphs. A representative example of these documents is shown in Figure 6.15.



Figure 6.15 Partial view of an XML AGG graph.

AGG uses basically four XML elements (see Figure 6.15):

- The elements *NodeType* and *EdgeType*: These elements allow us to specify which kind of nodes and edges can be defined in the graph. They provide support to store type graphs. These elements are defined by means of a name and a set of graphical representation properties. Furthermore, attributes can be included in the definition of types throughout the XML element *AttrType*.
- The elements *Node* and *Edge*: These elements allow us to define graphs. They are defined by means of a name, a set of attributes and a type. The type is a reference to a previously defined *NodeType* or *EdgeType* element. In this case,

6. Applying Traceability Rules through Model-to-Model Transformations

attributes can be also included in the definition of nodes throughout the XML element *Attribute*.

6.3 Automatic Application of Graph-Grammar-based Model-to-Model Transformations

We have presented above how traceability rules presented in Chapter 5 can be implemented as graph transformation rules. Furthermore, we have seen how the AGG environment provides us with graphical editors and transformation engines in order to visually define these graph transformation rules and automatically apply them. However, the fact that graph transformations can be automatically applied does not mean that traceability rules can be automatically applied into a task-based requirements model in order to obtain OOWS conceptual models. AGG allow us to automatically apply graph transformations in order to transform a graph into another graph. However, models are not graphs. Then, in order to automatically apply traceability rules by means of graph transformations:

- We must provide support for automatically obtaining a graph that represent the task-based requirements model. This graph is taken as host graph in the graph transformation process. This graph must be defined according to the type graph presented in this chapter (see Figure 6.4).
- Once transformation process is finished the resultant graph represents an OOWS conceptual model. Then, this graph must be automatically translated into the corresponding target model format.

In order to support these aspects we have developed the following two tools:

1. *Task2Graph*: This tool automatically obtains an AGG graph that represents a task-based requirements model. This tool takes as source an XMI document in which a task-based requirements model is stored by means of the RE tool presented in Chapter 4. Then, it applies a XML document transformation and obtains the equivalent graph defined according to the XML-based language used by AGG (see above).
2. *Graph2OOWS*: This tool obtains OOWS conceptual models from graph representations. The OOWS CASE tool uses XML-based documents for storing models (see Appendix B). Then, this tool also applies a XML document transformation in order to obtain the XML specification of an OOWS model from a XML-based document in which an AGG graph is stored.

6. Applying Traceability Rules through Model-to-Model Transformations

1. Task2Graph

Task2Graph is a tool that takes as source a XMI document in which a task-based requirements model is stored. As a result it obtains an XML document with the equivalent AGG graph.

Figure 6.16 shows a snapshot of the Task2graph user interface. It is a very intuitive interface divided in two main frames: (1) The Source File frame, which allows us to select the XMI document in which the task-based requirements model is stored. (2) The Target File frame, which allows us to indicate the file in which the AGG graph must be stored.

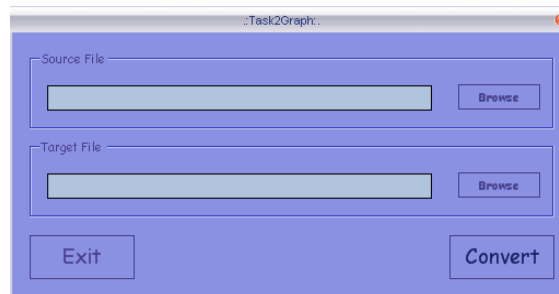


Figure 6.16 Task2Graph user interface

In order to generate the AGG XML document this tool creates first all the elements `NodeType` and `EdgeType` that are required. To do this, the type graph presented above is used. Next, elements `Node` and `Edge` are derived in order to define the graph that represents the task-based requirements model. This derivation is performed by following the next steps (see Figure 6.17):

- (1) The task taxonomy is represented as an AGG graph: tasks are derived into graph vertices, and refinement relationships into graph edges.
- (2) Each activity diagram is represented as an AGG graph: IPs and System Actions are derived into graph vertices whereas the activity diagram's arcs into graph edges.
- (3) These defined AGG graphs are joined into a single graph: each vertex that represents an elementary task is connected to the vertices that constitute its initial and final step.
- (4) Information templates are then represented as AGG graphs: First, a graph vertex is defined to represent each information template. For each entity feature defined in the template, we define a vertex that is connected to the template node.

6. Applying Traceability Rules through Model-to-Model Transformations

- (5) Finally, vertices that represent simple nature entity features are connected to the interaction points in which they are shown (Output IP) or requested (Input IP). To do this, nodes for representing concepts such as ExchangedDataTemplate, ExchangedDataSection and ExchangedElement are used.

The AGG graph shown in Figure 6.17 is a partial version of the graph that represents the task-based requirement model presented in Chapter 4.

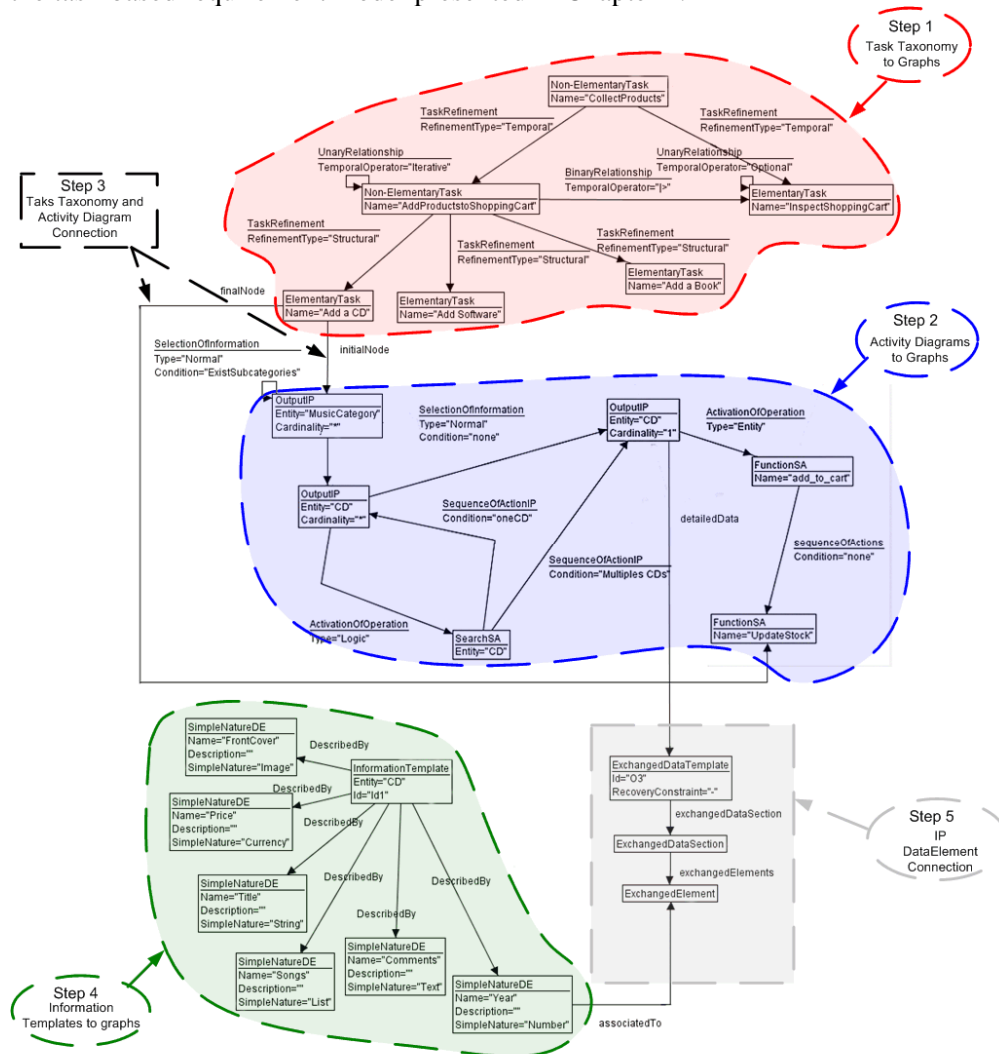


Figure 6.17 Graph representation of a task-based requirements model

6. Applying Traceability Rules through Model-to-Model Transformations

2. Graph2OOWS

Graph2OOWS is a tool that allows us to automatically obtain XML specifications of OOWS conceptual models from AGG graphs.

When the AGG tool finishes the graph transformation, the host graph is extended with two graphs that represent both an OOWS navigational models and an OOWS structural model. These two graphs constitute the source of the Graph2OOWS tool (stored in a XML-based AGG document).

Figure 6.18 and Figure 6.19 show two examples of the graphs that constitute the source of the Graph2OOWS tool. Figure 6.18 shows a graph that represents a part of the OOWS structural model that is presented in Chapter 5 (see Figure 5.13). In particular, this graph describes three classes (Music Category, CD and Artist) with its attributes, operations and the relationships between these classes. Figure 6.19 shows a graph that represents part of the OOWS navigational model that is presented in Chapter 5 (see Figure 5.31). This graph represents three navigational contexts (Music Category, CD and Artist). For each navigational context its manager navigational class with its navigational attributes and operations are also represented. For the context CD we can see how an index and a filter are defined. Finally, notice how navigational contexts are connected (Music Category with CD and CD with Artist) throughout the definition of context navigational relationships.

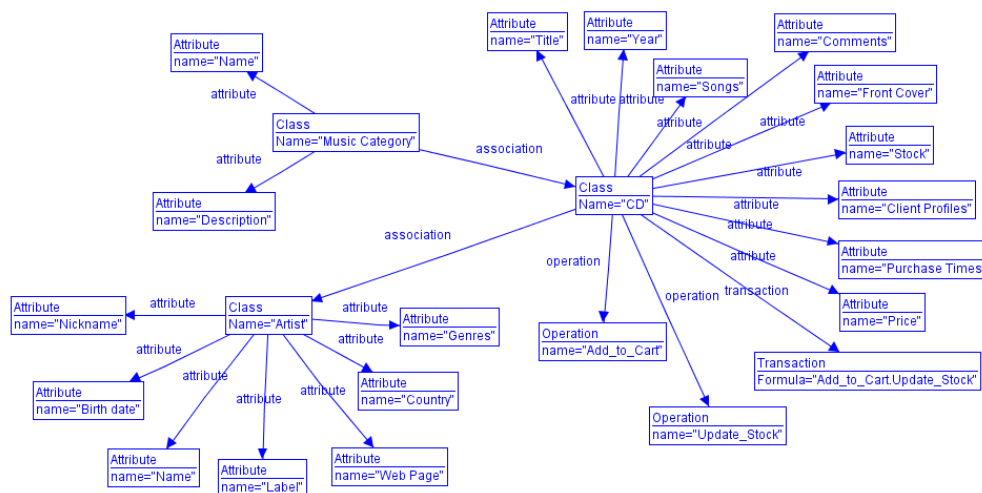


Figure 6.18 Graph that represents an OOWS Structural Model

6. Applying Traceability Rules through Model-to-Model Transformations

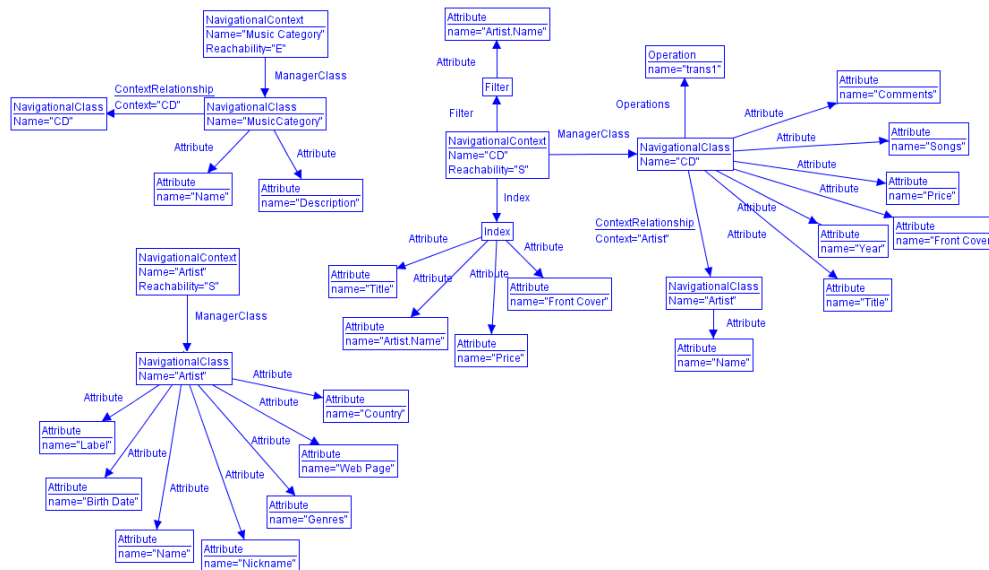


Figure 6.19 Graph that represents an OOWS Navigational Model

Figure 6.20 shows a snapshot of the Graph2OOWS user interface. It is a very intuitive interface divided in two main frames: (1) The *Source File* frame, which allows us to select the XML document where the AGG graph that represent an OOWS model is stored. (2) The *Target File* frame, which allows us to indicate the file in which we store the OOWS model in the proper format.

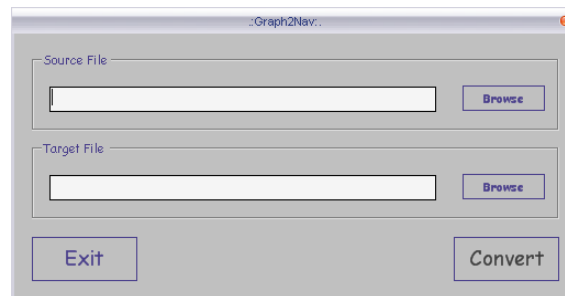


Figure 6.20 Graph2OOWS user interface

When an AGG document is loaded in the Graph2OOWS tool, the tool reads the node types defined in the document. From this list of node types, the tool identifies those that represent OOWS conceptual primitives. Then, the tool looks for these types of nodes in the document. For each of these nodes, the tool generates the corresponding OOWS XML specification. Then, this XML specification can be loaded in the OOWS case tool. Figure 6.21 shows the part of the navigational model that we obtain

6. Applying Traceability Rules through Model-to-Model Transformations

when the XML specification obtained from the graph in Figure 6.19 is loaded in the OOWS CASE tool.

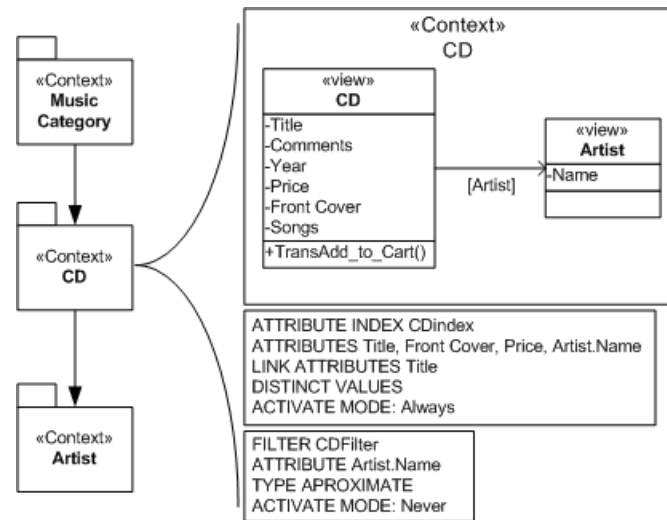


Figure 6.21 Navigational model derived from a graph-based description

3. The tool supported model-to-model transformation strategy

Next, we present the big picture of the tool-supported strategy for automating the model-to-model transformations. This strategy is graphically shown in Figure 6.22.

The proposed strategy takes as source task-based requirements model stored in XMI documents. These documents are obtained by means of the RE tool presented in Chapter 4. These documents are loaded into the Task2Graph tool which obtains the equivalent AGG graph. This graph is next loaded into the AGG tool. This tool applies the set of graph transformation rules that implements the traceability rules shown in Chapter 5. After the graph transformation, the host graph is extended with two graphs that represent an OOWS structural model and an OOWS navigational model. This extended graph is loaded in the Graph2OOWS tool in order to obtain the equivalent OOWS model in the proper XML format. This document can be loaded in the OOWS CASE tool in order to obtain the conceptual models.

6. Applying Traceability Rules through Model-to-Model Transformations

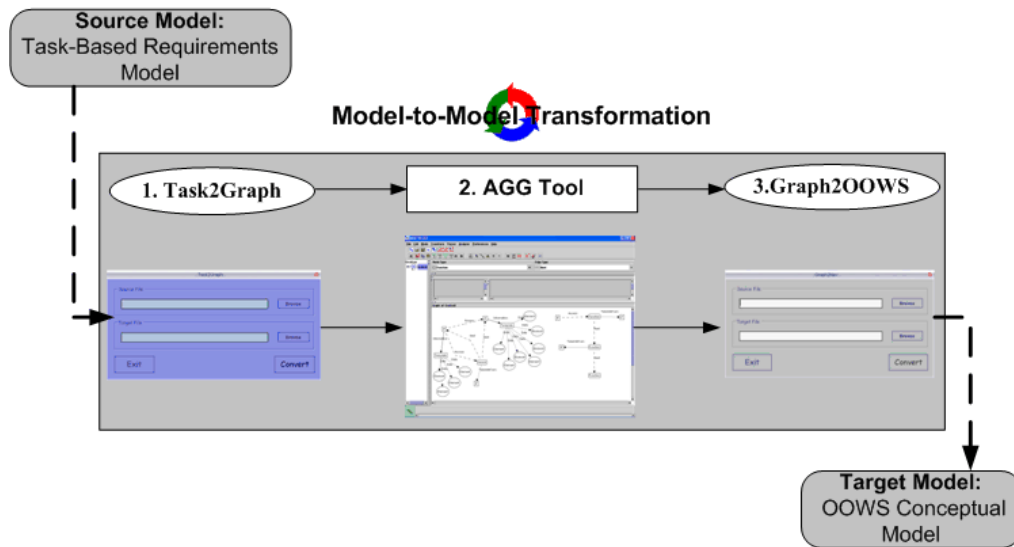


Figure 6.22 Tool-supported strategy for automating model-to-model transformations

6.4 Conclusions

In this chapter, we have presented a technique that allows us to automatically apply the traceability rules presented in Chapter 5. This technique is based on the use of graph grammars. A graph grammar is made up of a host graph (that in this case represent a task-based requirements model) and a transformation system. The transformation system is made up of a set of graph transformation rules that transforms the host graph into a target graph (in this case, an OOWS conceptual model). We have defined two graph grammars in order to derive two OOWS conceptual models (the structural and navigational models) from task-based requirements models. These two graph grammars can be considered to be two model-to-model transformations between task-based requirements models and (1) OOWS structural models and (2) OOWS navigational models.

In order to apply these two model-to-model transformations automatically a set of tools are proposed: (1) the AGG system is used to automatically apply the graph transformation rules; (2) the Task2Graph and Graph2OOWS tools are used to transform the source model into a graph and the resultant graph into a model, respectively.

The use of graph transformations as a model transformation technique provides us with benefits such as a well-known underlying formalism in which researchers have been working during decades, the possibility of defining transformations in a visual way or a wide range of open-source tools that we can use for creating the graph

6. Applying Traceability Rules through Model-to-Model Transformations

transformations. In this context, we want to emphasize this last benefit because, although currently we can find several tools for defining model-to-model transformations (such as ATL or QVT) this was not the situation when the work of this thesis starts.

Furthermore, it is worth to remark that, although the main ideas proposed in this chapter can be generalized for performing any model-to-model transformation (i.e. the use of XML transformations combined with AGG), they have been defined as an implementation of the traceability rules presented in Chapter 5. In this context, this technique can be considered to be a particular implementation of these traceability rules. However, this is not the only valid implementation for these rules. In fact, we are currently working on implementing them by using other technologies that are arising for performing Model-to-Model transformations (such as ATL or QVT). However, the use of these new technologies constitutes only this, a change of technology. The transformations that are being defined by using these technologies are based on the same traceability rules that have been defined in Chapter 5.

Chapter 7

*“If the only tool you have is a hammer,
you tend to see every problem as a nail.”*

Abraham Maslow
(American psychologist, 1908-1970)

7 Prototyping Requirements

One of the main contributions of this thesis is the definition of a Web application RE process that encourages the use of prototypes in RE activities (see Chapter 3). This process proposes the generation of Web application prototypes from requirements specifications in order to be used for validating requirements.

In order to generate these prototypes several tools are used along the RE process: the RE tool presented in Chapter 4 for specifying requirements, the set of tools presented in chapter 6 for automatically performing model-to-model transformations and finally the OOWS CASE tool presented in Appendix B to generate code.

In order to better understand how these tools interoperate among them a “big picture” of the use of them is presented in this chapter. To do this, we show again the RE process presented in Chapter 3, locating in each activity the tools that have been presented throughout this thesis (Section 7.1). Furthermore, we analyze the results obtained by each tool when they are used to generate a Web application prototype for the Amazon example (Section 7.2). In this analysis, we pay special attention to the generated prototype, studying moreover how this product software satisfies the user’s needs specified in the task-based requirements model (Section 7.3). Finally, we conclude this chapter in Section 7.4.

7.1 Tool Support: A “Big Picture”

The generation of prototypes is proposed to be performed by using several tools along the different activities inside the RE process. This process is divided in three activities: requirements elicitation, requirements specification and requirements validation. As we have explained in Chapter 3, the efforts of the work of this thesis are focused on improving the last two activities (requirements specification and validation). Thus, each of the tools presented in this thesis is used in either the requirements specification activity or the requirements validation activity. These tools must cooperatively interoperate among them in order to properly generate the Web application prototype. This interoperation is shown in Figure 7.1 where the RE process together with the tools used in each activity is presented.

First, the RE tool presented in Chapter 4 is used in the requirements specification activity. This tool allows us to create task-based requirements models. These models are stored in a XMI documents.

The other tools are used for obtaining Web application prototypes from the requirements model. Then, they are used in the validation of requirements. In order to obtain Web application prototypes directly from task-based requirements models, the model-to-model transformation strategy presented in Chapter 6 together with the OOWS code generation strategy presented in Appendix B are used:

- Model-to-model transformations are performed by means of three tools: Task2Graph, AGG and Graph2OOWS. The first tool takes as source a XMI document created by the RE tool (where a task-based requirements model is stored) and transforms it into the corresponding AGG graph. Then, AGG transforms this graph into other graphs that represent OOWS conceptual models. Finally, the Graph2OOWS tool transforms these graphs into the proper OOWS XML specification.
- The OOWS code generation strategy is implemented by means of the OOWS CASE tool presented in Appendix B. This tool take as source XML documents where OOWS conceptual models are specified and transforms them into the corresponding Web application prototype.

7. Prototyping Requirements

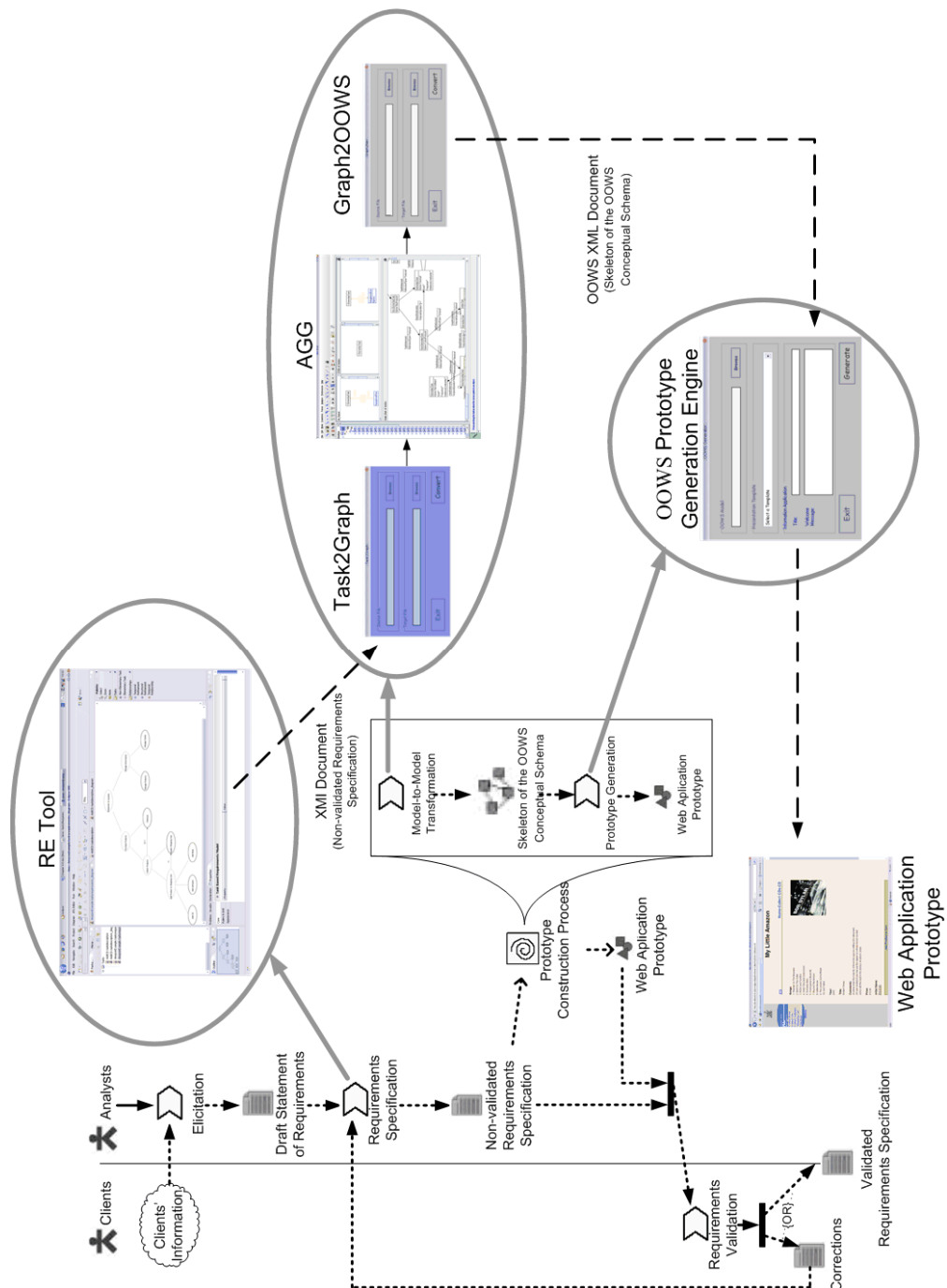


Figure 7.1 Tool Support in RE activities

7.2 From Requirements to Prototypes. Step by Step.

According to Section 7.1, the different steps that are performed in order to generate prototypes from requirements are the following:

1. To create a task-based requirements model. To do this, we use the RE tool.
2. To represent the task-based requirements model as an AGG graph. To do this, we use the Task2AGG tool.
3. To transform the graph. To do this, we use the AGG tool.
4. To translate the obtained graph into the proper OOWS XML format. To do this, we use the AGG2OOWS tool.
5. To generate code. To do this, we use the OOWS CASE tool.

Next, we analyze the results obtained in each step when the corresponding tool is used for the Amazon example. We also explain how the result obtained in each step is taken as source for the next step. A more complete description of this analysis can be found in Appendix D.

7.2.1 Step 1: A Task-based Requirements Model

The result that is obtained in the first step is a task-based requirements model. This model is created by the RE tool presented in Chapter 4. It is stored in a XMI document. A brief description of the requirements model for the Amazon example has been already introduced in Chapter 4. Figure 7.2A shows, as a representative example, the task performance description of the elementary task Add CD, defined in the RE tool. Figure 7.2B shows a partial view of the *amazon.xmi* document, where the task-based requirements model is stored.

7. Prototyping Requirements

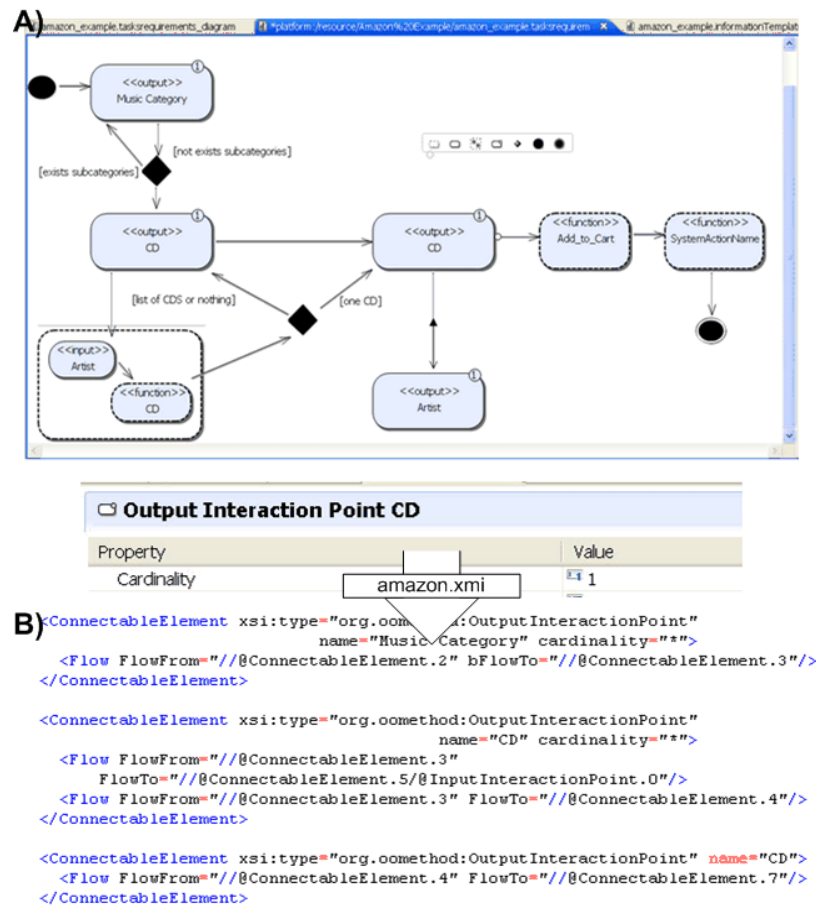


Figure 7.2 Step 1: Creation of the task-based requirements model

7.2.2 Step 2: A Graph-based Representation of the Task-based Requirements Model

The second step consists in obtaining a graph representation of the task-based requirements model. We use the Task2Graph tool. This tool has been introduced in Chapter 6. Figure 7.3A shows how the *amazon.xmi* file that is obtained in the previous step is introduced as source in the Task2Graph tool and how the *amazongraph.ggx* file is defined as target. Figure 7.3B shows a partial view of the graph that is created in the *amazongraph.ggx* file. In particular, this figure shows the part of the graph that represents the task performance description of the elementary task Add CD.

7. Prototyping Requirements

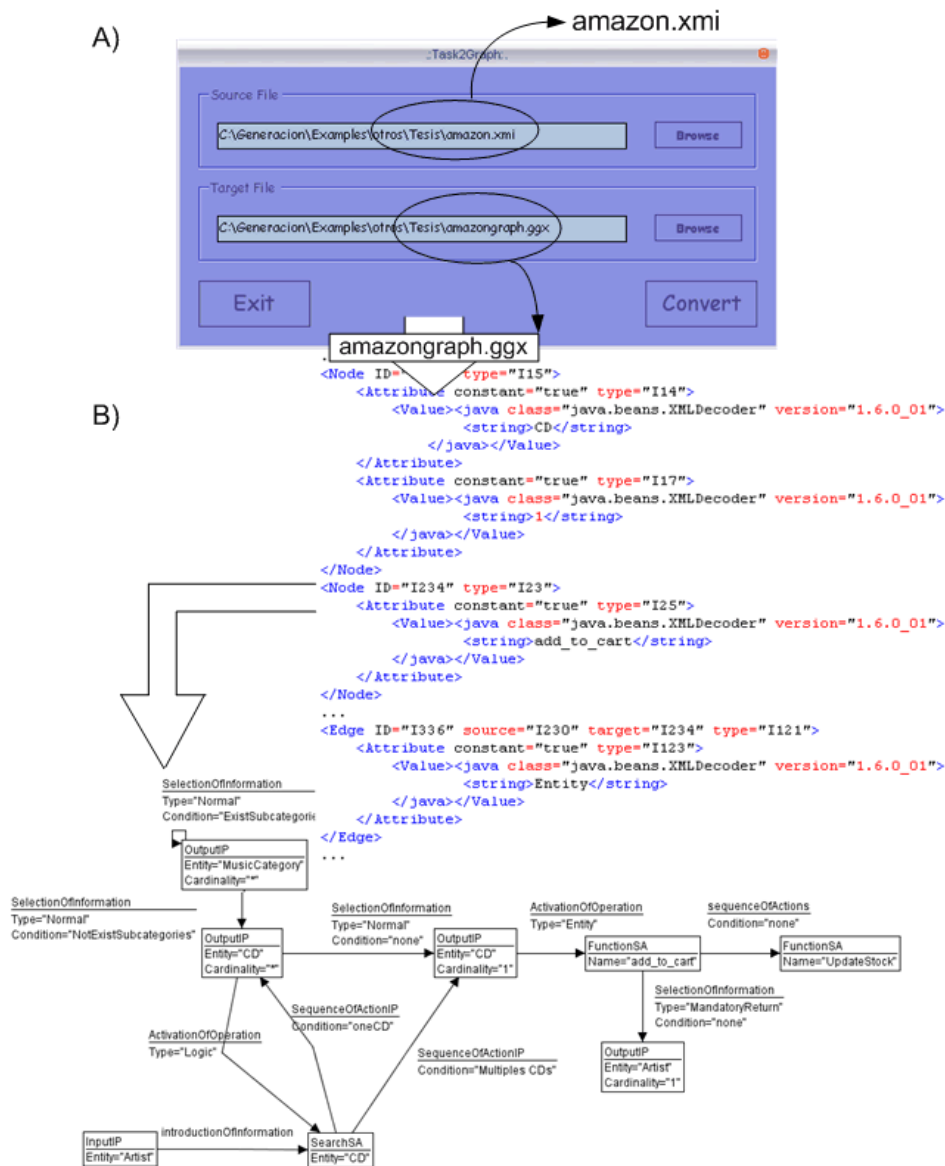


Figure 7.3 Step 2: Task-based requirements models as graphs

7.2.3 Step 3: Transforming the Graph

The third step consists in transforming the AGG graph obtained in the previous step into other AGG graphs that represent OOWS models. We use the AGG tool. This tool has been introduced in Chapter 6. Figure 7.4A shows how the *amazongraph.ggx* file is introduced as source in the AGG tool. Figure

7. Prototyping Requirements

7.4B shows how the same file contains a graph that represent an OOWS navigational model after the transformation. The partial graph that is shown represents the navigational context CD.

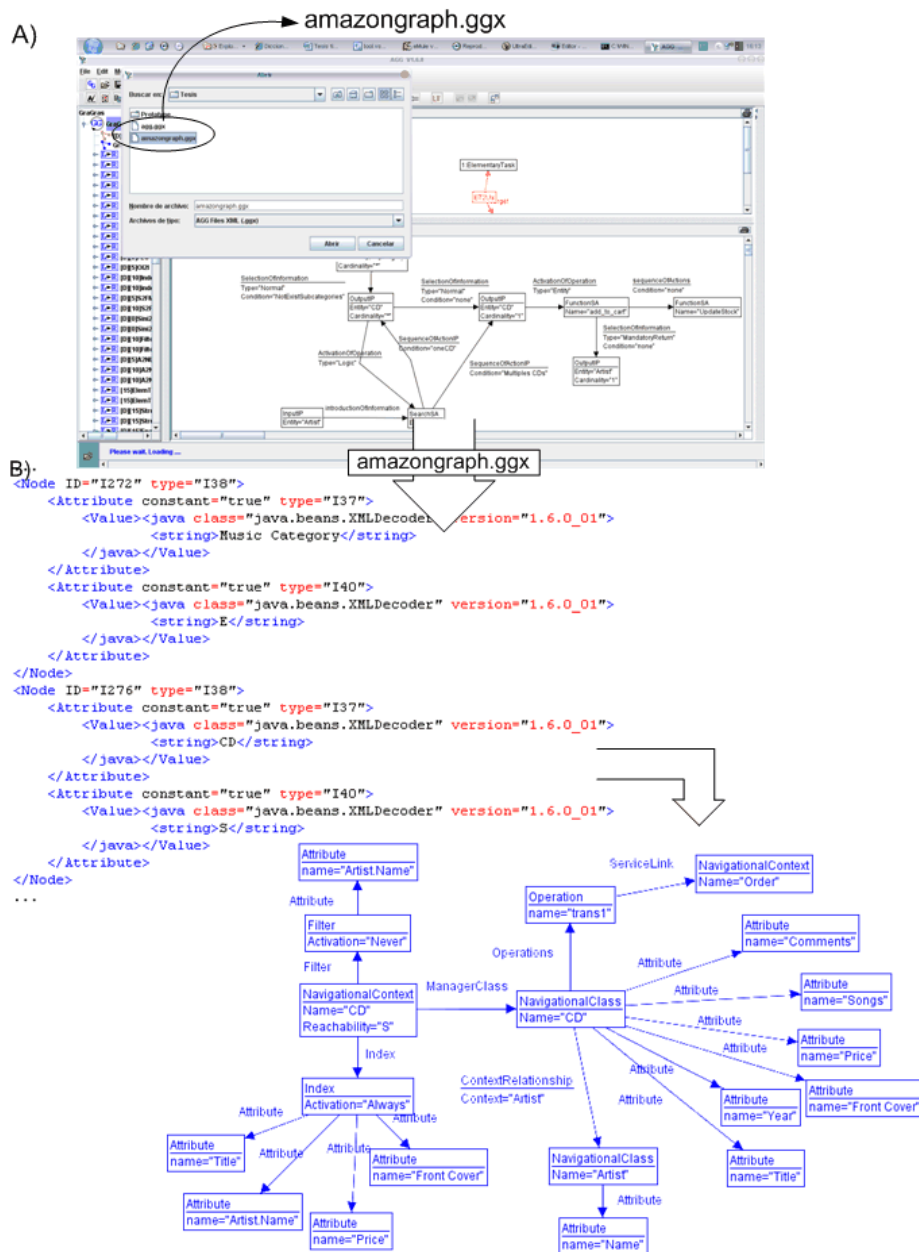


Figure 7.4 Step 3: OOWS models as graphs

7. Prototyping Requirements

7.2.4 Step 4: OOWS Models in the Proper XML Specification

The forth step consists in transforming the AGG graph obtained in the previous step into the proper OOWS XML specification. We use the Graph2OOWS tool. This tool has been introduced in Chapter 6. Figure 7.5A shows how the *amazongraph.ggx* file is introduced as source into the Graph2OOWS tool and how the file *oowsmode.xml* is defined as target. Figure 7.5B shows a partial view of the model that is created in the *oowsmode.xml* file. In particular, we show the navigational context CD.

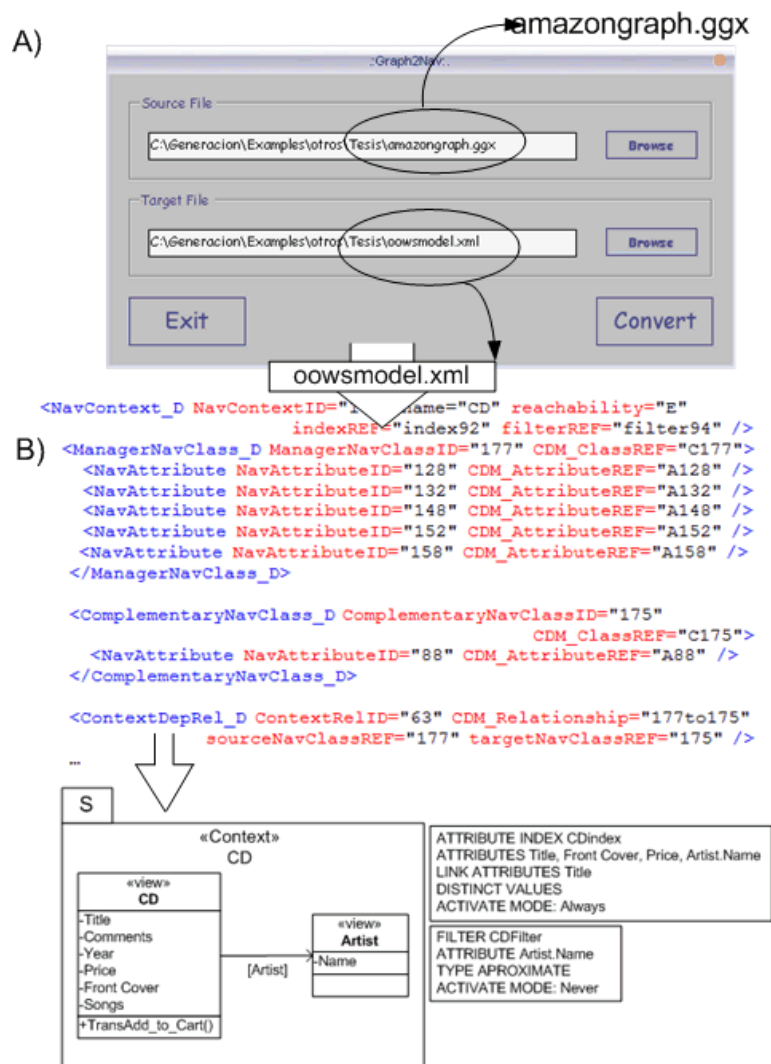


Figure 7.5 Step 4: OOWS models in the proper format

7. Prototyping Requirements

7.2.5 Step 5: Generation of Web Application Prototypes

The fifth step consists in generating a Web application prototype from the OOWS XML-based specification obtained in the previous step. We use the OOWS CASE tool. This tool is introduced in Appendix B. Figure 7.6 shows how the *oowsmodel.xml* file is introduced as source into the OOWS prototype generation engine. In order to generate a prototype we indicate a default presentation template (dsic). Furthermore, a title (“My Little Amazon”) and a welcome message (“You can start to buy products by selecting the different options at the left side menu”) are also introduced.

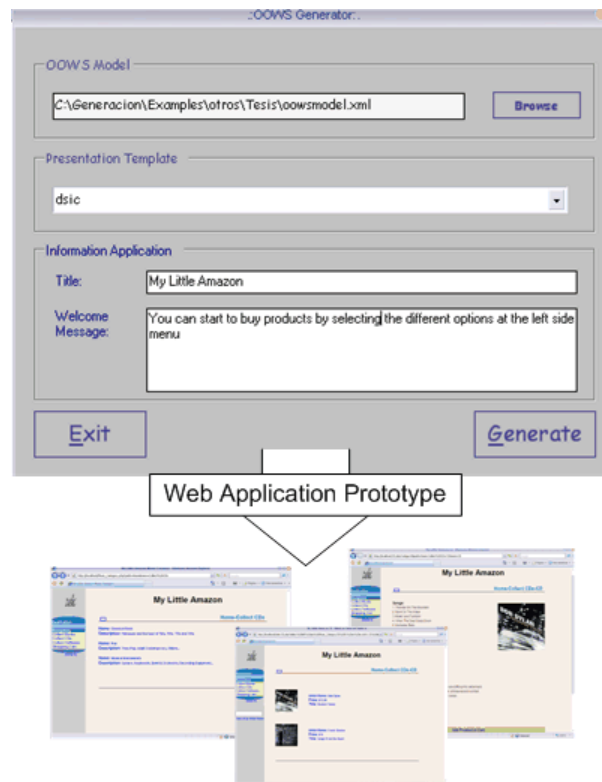


Figure 7.6 Step 5: Web application prototype generation

The Web application prototype that is generated is divided into two main folders (see Appendix B) that contain (1) a SQL script for creating the database in which we store the information that the Web application prototype must support (see an example in Appendix D) and (2) a set of PHP files that implements the navigational structure and the access to the database.

7. Prototyping Requirements

The PHP files that implement the navigational structure are generated from the different navigational contexts defined in the navigational model as well as from the different access mechanisms (indexes and filters, see Appendix B). Figure 7.7 shows the different files that have been generated for supporting part of the Amazon Example.

These PHP files provide the user with the information that supports the abstract mechanisms that they implements. For instance, if a PHP file implements a navigational context it provides the information defined in the view of the navigational context; if a PHP file implements an index it provides a list defined from the attributes associated to the index. In order to retrieval this information SQL queries are defined.

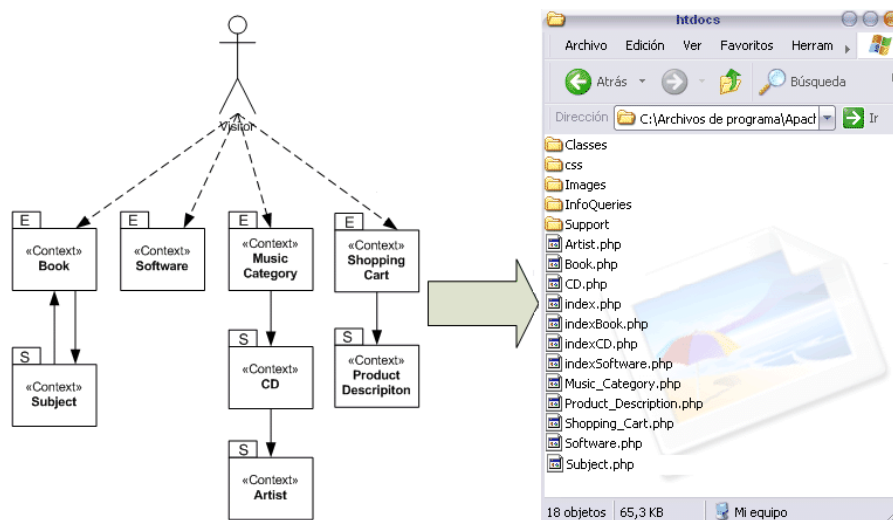


Figure 7.7 Generated files for the Amazon example

Furthermore, these PHP files also provide users with the links that allow them to properly navigate the different pages. These links are created from the different exploration and sequence navigational links defined in the navigational model. Exploration links are used to define the main menu of the Web application prototype (which is usually located at the left side of each Web page). Sequence links are used to create links inside the information provided by each Web page. Figure 7.8 shows how the different PHP files implement the different navigational contexts and how they are interconnected according to the defined navigational links. Notices how navigational contexts with an index are implemented with two PHP files (one for the index and another for the context definition).

7. Prototyping Requirements

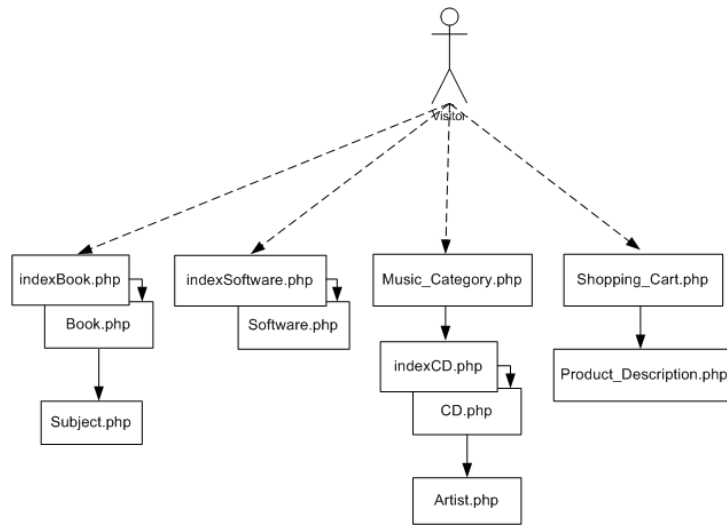


Figure 7.8 PHP files interconnected according to the navigational model

In the next section, we can see some examples of the Web pages that are produced by these PHP files. In particular: the page produced by the file `Music_Category.php` is shown in Figure 7.10A; the page produced by the file `indexCD.php` is shown in Figure 7.10B; the page produced by the file `CD.php` is shown in Figure 7.11A; and the page produced by the file `Artist.php` is shown in Figure 7.11B.

7.3 Supporting Requirements with the Generated Web Application Prototypes

In this section, we analyze whether the generated Web application prototype correctly support the requirements specified for the Amazon example. To do this, we compare some parts of the task-based requirements model presented in Chapter 4 with the Web pages that support them. In particular, we focus this study on how the requirements related to the collection of products are supported.

7.3.1 Supporting the Task Taxonomy

Requirements Model: According to the task taxonomy presented in Chapter 4 (see Figure 4.5), when users access the system they can collect three type of products: CDs (elementary task Add CD), software products (elementary task Add Software) and books (elementary task Add Book). Furthermore, according to the temporal relationship defined in the task taxonomy users must be able to inspect their shopping cart when they are collecting products.

7. Prototyping Requirements

Web Application Prototype: Figure 7.9 shows the generated home page that is accessed by users when they connect to the Web application. As we can see in the main menu, this page allows users to collect CDs, software products and books. Furthermore, we can see how the shopping cart is also available from the main menu. The main menu is available from every Web page that implements the prototype. In this context, users have always the possibility of inspecting the shopping cart when they are collection products.

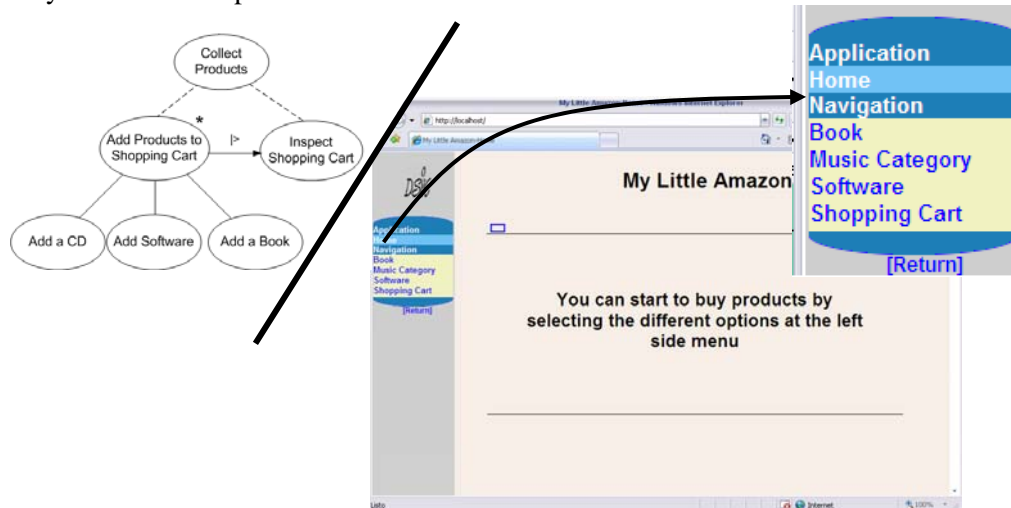


Figure 7.9 Home page for the Amazon Example

We can see in Figure 7.9 how the different tasks associated to the collect of products are available in the Web application prototype. Next, we analyze whether the generated Web pages allow users to perform these tasks according to their description. To do this, we focus on the pages that allow users to add CDs to the shopping cart. The other pages are omitted to not overload this chapter. They are however implemented in analogous way.

7.3.2 Adding CDs: Lists of Music Categories and CDs

Requirements Model: According to the performance description of the elementary task Add CD, users access first a list of music categories (see Figure 4.20). For each music category the name and the description is shown (the exchanged data template can be found in Figure D16 of Appendix D).

Web Application Prototype: Figure 7.10 shows the Web page that users access when they click over the entry Collect CDs of the main menu. This page provides users with a list of music categories. For each music category the name and the description is shown.

7. Prototyping Requirements

Requirements Model: From the list of music categories users can select one in order to access either the list of CDs that belong to it or a list of subcategories (see Figure 4.20). In the list of CDs, users can obtain the title, the price, the artist name and the front cover of each CD (see exchanged data template in Figure 4.26).

Web Application Prototype: When users click over a music category in the page presented in Figure 7.10A they access the Web page that is shown in Figure 7.10B. This page shows a list of CDs. For each CD the title, the price, the artist name and the front cover are shown. The access to a list of subcategories is not supported by the prototype because this characteristic cannot be represented by the conceptual primitives provided by the OOWS method.

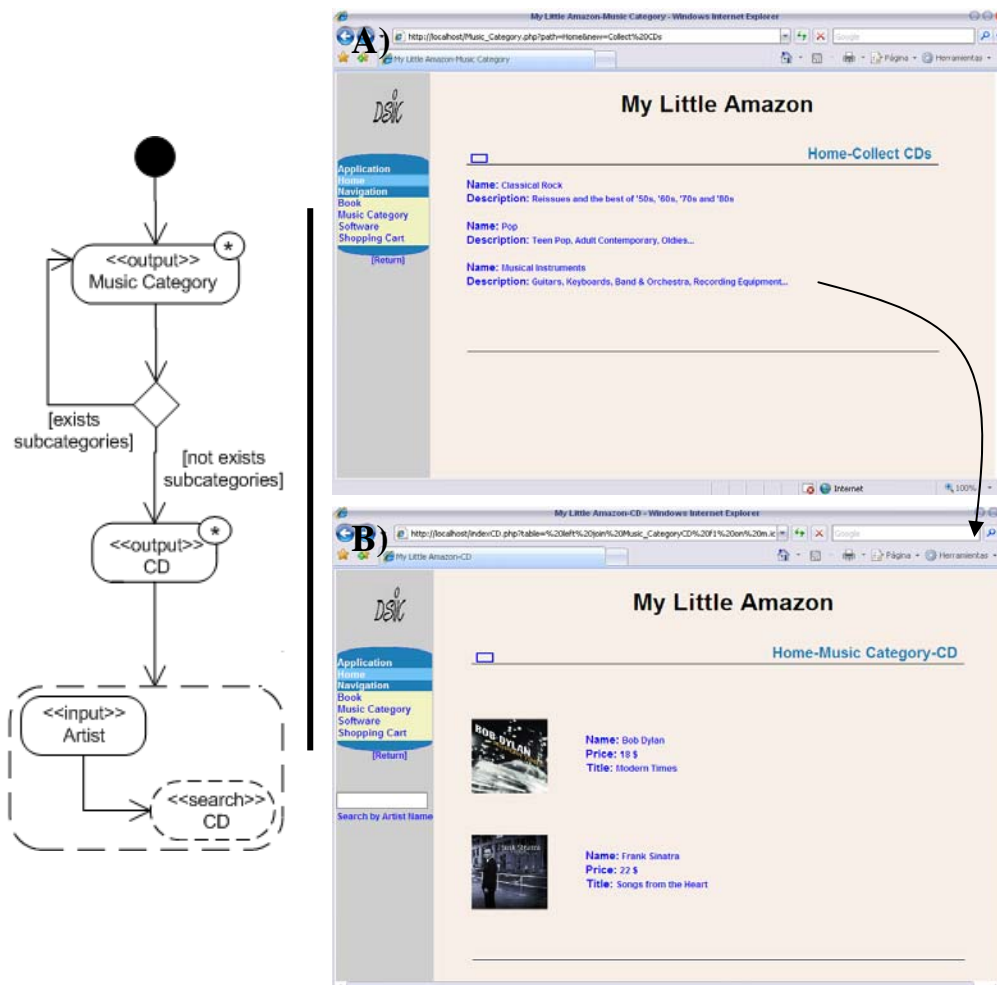


Figure 7.10 Web pages that provide lists of Music Categories and CDs

7. Prototyping Requirements

Requirements Model: From the list of CDs, users can perform two actions (see Figure 4.20): (1) to activate a search system action in order to search CDs from the name of an artist (the corresponding exchanged data template can be found in Figure D20 of Appendix D), or (2) select a CD in order to access a detailed description of it.

Web Application Prototype: The Web page that provides user with a list of CDs (see Figure 7.10B) implements a search engine that allows them to search CDs from the name of an artist (see left side of this page). Furthermore, when users click over a CD of this list they access the Web page that is shown in Figure 7.11A where a detailed description of the CD is provided

7.3.3 Adding CDs: Descriptions of CDs and Artists

Requirements Model: According to the exchanged data template associated to the IP Output (CD,1) (see Figure D18 of Appendix D), when users access the detailed description of a CD they obtain the title of the CD, the price, the front cover, the artist name, the year, the songs and some comments.

Web Application Prototype: The Web page that is shown in Figure 7.11A implements this detailed description. As we can see, all the above-introduced information is shown: the title of the CD, the price, the front cover, the artist name, the year, the songs and some comments.

Requirements Model: Once users have accessed the description of a CD they can finish the task by adding this CD to the shopping cart (see Figure 4.20). Additionally, before adding the CD, users can access detailed information about the artist. This detailed information is defined from the name of the artist, the nick name, the birth date, the country where the artist was born, the label that recorded the CD and the Web page of the artist (the corresponding exchanged data template can be found in Figure D19 of Appendix D).

Web Application Prototype: The description of a CD is provided by the Web page in Figure 7.11A. We can see how in the bottom side of this page users can add to CD to the shopping cart. Furthermore, the name of the artist is implemented as a link. When users click over this link they access the Web page in Figure 7.11B. This page provides detailed information about the artist: the name, the nick name, the birth date, the country, the label and the Web page.

Finally, notice how none of the presented Web pages provide the information by following a specific order. For instance, the Web page that is shown in Figure 7.11A presents first the list of songs than the title of the CD (which differs from the way in which a CD description is usually provided by E-commerce applications). In the same way, the Web page that is shown in Figure 7.11B presents first the birth date of the artist than the artist name. The reason of this is that Web pages have been automatically generated from the OOWS conceptual models obtained by applying graph transformations. The graph-based representations that are handled during the

7. Prototyping Requirements

transformation process just indicate the data that must be shown to users without considering presentation aspects such as data order.

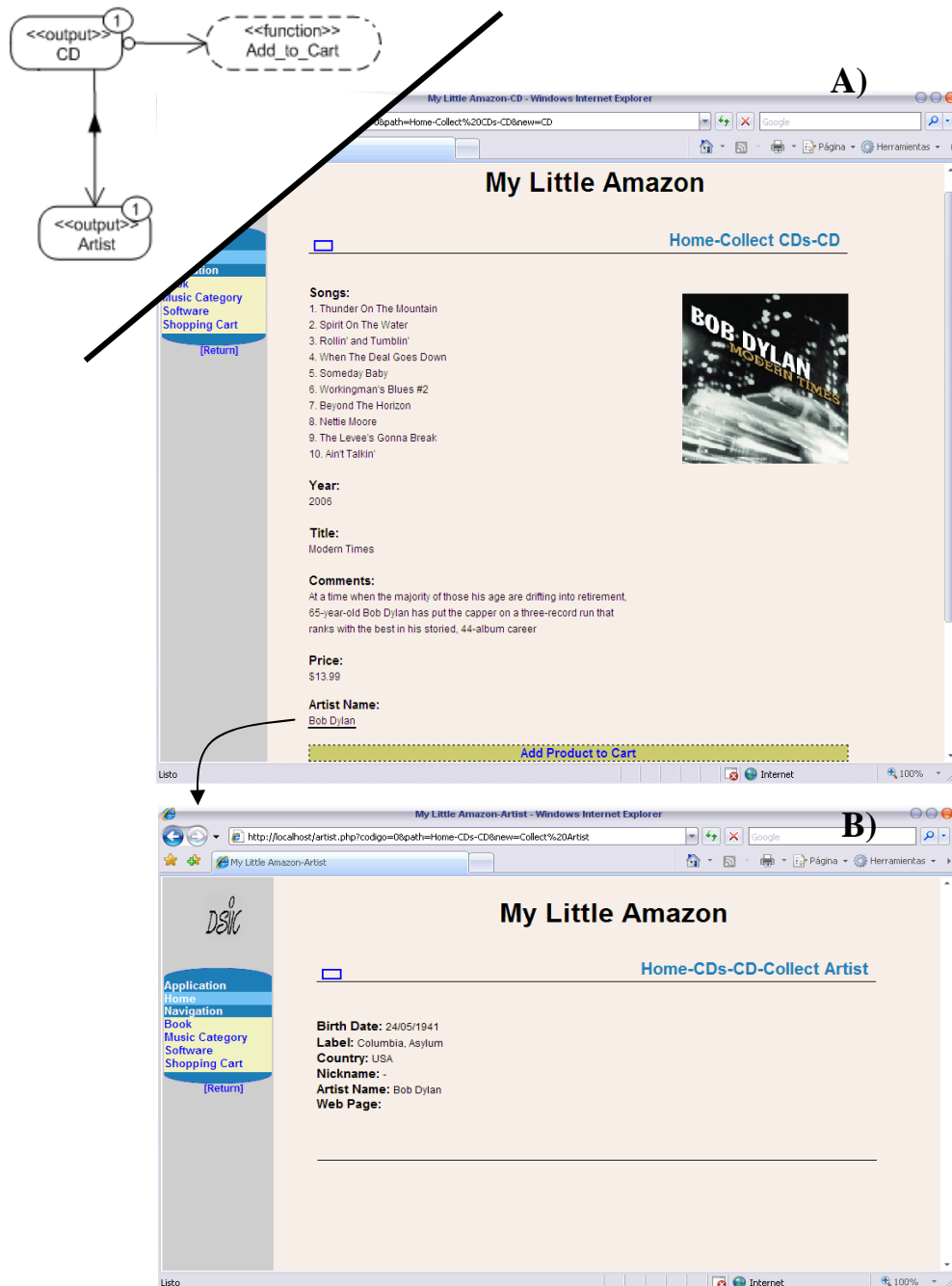


Figure 7.11 Web pages that provide descriptions about a CD and an artist

7.4 Conclusions

In this chapter, we have explained how the different tools presented in this thesis are used in order to generate Web application prototypes. We have analyze how they interoperate by indicating which results they produce and how these results are handled by the other tools. Furthermore, we have also indicated in which activities of the RE process these tools are used.

Additionally, we have presented the Web application prototypes that are generated by using the OOWS code generation capabilities. These prototypes are implemented as a set of interconnected PHP files that are in charge of implementing the navigational structure of the Web application. These files also access the database in order to extract the information that must be shown to users.

Finally, we have analyzed how the set of PHP files supports the requirements specified in the task-based requirements model. We have presented the set of Web pages that is produced by the PHP files in order to allow users to collect CDs. Next, we have study how the different requirements related to the collection of CDs (described throughout the task taxonomy and the elementary task Add CD) are supported by these Web pages.

Chapter 8

“This is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning”

Winston Churchill
(British Orator, Author and Prime Minister during World War II, 1874-1965)

8 Conclusions

In this last chapter, we introduce the conclusions of the work presented in this thesis. First, we list the main contributions of this thesis in the area of requirements engineering for Web applications. Next, we explain the work that is currently being performed as well as future work. Finally, the publications that have been obtained from the work of this thesis are cited.

8.1 Main Contributions

We have stated that the work of this thesis is mainly related to two engineering paradigms: Requirements Engineering (RE) and Web Engineering (WE). This work is related to RE because we have presented a new RE approach for handling requirements. It is related to WE because this approach is focused on improving RE activities in the development of Web applications.

Additionally, we can also relate this work to the field of model driven-development as well as to the area of rapid prototyping. On the one hand, the RE approach presented

8. Conclusions

in this thesis has been used to extend the model-driven development process of the OOWS Web engineering method. On the other hand, the work of this thesis constitutes a key factor in the definition of a strategy to automatically obtain Web application prototypes from requirements.

In this context, we can state that the main contributions that we have introduced are the following:

- 1 The development of a task-based requirements model for specifying Web applications requirements. This model allows us to properly consider distinctive characteristics of Web applications such as Navigation at the requirements level. To do this, we have introduced mechanisms for creating a task taxonomy where the different tasks that users must perform by interacting with a Web application are represented. Furthermore, a novel technique based on activity diagrams has been introduced to describe the performance of tasks from the interaction between the user and the system. Finally, these descriptions are complemented with information templates in order to consider informational requirements.
- 2 The development of an RE tool for supporting the visual creation and the management of task-based requirements models. To develop this tool we have used a technology that is based on the Eclipse platform.
- 3 The definition of a particular interpretation of the task-based requirements model based on a set of traceability rules. These rules guide analysts in the creation of OOWS conceptual models that satisfy the requirements captured in the task-based requirements model (which improves the traditional problem stated as *how to go from the problem space to the solution space*). Furthermore, these rules also constitute a valuable mechanism for tracing the mistakes detected in the conceptual model to the requirements model.
- 4 The implementation of the traceability rules by using a technique based on graph transformations. This technique, which is supported by tools, allows us to define model-to-model transformations that automatically obtain OOWS conceptual models from task-based requirements models.
- 5 The incorporation of the RE approach presented in this thesis into the development process of the OOWS Web engineering method. To do this, we have described the OOWS development process with a proper notation such as SPEM, and we have introduced a new initial stage in which requirements are handled.
- 6 The definition of an iterative and prototyping RE process for Web applications to properly support the new stage introduced in the OOWS development process. This RE process proposes the iterative performance of different

8. Conclusions

activities in order to create the task-based requirements model. Within these activities, Web application prototypes are automatically generated from requirements models. These prototypes constitute a valuable tool for helping customers to validate requirements. To obtain these prototypes, both the tool-supported technique for applying traceability rules and the OOWS strategy of automatic code generation are used.

- 7 The validation of the RE approach by applying it in the development of a case study. This case study can be found in Appendix D. In this appendix, we have presented a complete requirements model of an E-commerce application and the Web application prototype that is automatically obtained from it.

Furthermore, the work of this thesis has been validated through its presentation in several seminars, where different Phd students have used it to resolve several case studies. The seminars in which this work has been presented are the following:

WEE-NET: Web Engineering Network of Excellence
Summer School in La Plata,
29th January – 9th February, 2007

SISCOM: Seminario Internacional en Sistemas y Computación
Fundación Universitaria Juan de Castellanos
Tunja – Colombia
26th and 27th October, 2007

8.2 Current and Further Work

This thesis is not a closed work and many research efforts can still be done in this research area. Many research activities are currently underway, and further research is ongoing in different and complementary directions.

The main research activities that are currently being performed are:

- The improvement of the traceability rule catalogue. As explained in Chapter 5, the proposed rule catalogue is not a closed catalogue. We are currently using and refining the proposed rules in the development of case studies. Furthermore, the OOWS method is also not a closed research topic. In this context, improved versions of OOWS are continuously being developed. Thus, new traceability rules must be defined if new conceptual primitives are introduced by these improved versions.

8. Conclusions

- The graphical editor is being tested and improved. We are currently using the graphical editor in the creation of several task-based requirements model in order to detect and correct possible mistakes. We are also refining the user interface in order to provide a more usable interaction experience.
- Migration of transformation technology. We are currently implementing the traceability rules by using ATL. This aspect will allow us to provide a more integrated RE environment where the graphical editor and the model transformation engine constitute a single tool.
- Elicitation Requirements. The identification of user tasks is not always easy and many times require analysts to have expertise in the use of this technique. In order to solve this problem, we are currently working on a technique for eliciting requirements. This technique uses requirements ontologies to help analysts in the identification of tasks.
- End user development. The strategy to automatically generate Web application prototypes together with techniques for eliciting requirements based on ontologies can be complemented with tools for end-users. These tools are designed to be used by people without programming knowledge. The idea is that by following a guided question process, end-users can provide these tools with enough information to create a task-based requirements model. Then, a Web application prototype can be generated from this model.
- Adaptive Requirements. We are also working on extending the task-based requirements model in order to capture adaptive requirements of Web applications. Currently, we are working on both a diagram to capture user profiles and mechanisms to define restrictions and preconditions from characteristics of the user profiles.

Immediate research activities include:

- The definition of a multidisciplinary RE process. The development of a Web application not only involves computer analysts but also other professionals such as graphic designers, usability experts, lawyers, etc. In this context, RE activities must properly consider the role that each of these professionals plays in providing techniques and tools that allow them to work in a cooperative way.
- A tool for supporting traceability aspects. As explained above, traceability rules constitute a valuable mechanism to trace mistakes in the different development artifacts. We want to develop a tool that helps analysts to perform this trace.
- Presentation requirements. Aspects related to presentation are not captured in the proposed RE approach. A forthcoming work will focus on complementary

8. Conclusions

task descriptions with sketches that allow us to capture presentation requirements.

8.3 Publications

The work related to this thesis has been published in two international journals, one book chapter, nine international conferences and one international workshop.

International Journals:

- Lorna Uden, **Pedro Valderas** and Oscar Pastor. *Activity Theory for analyzing Web application Requirements*. Information research - an international electronic journal. JCR 0.87. Accepted, awaiting revision and publication.

Citations: -

- **Pedro Valderas**, Vicente Pelechano, Oscar Pastor. *A Transformational Approach to Produce Web Application Prototypes from a Web Requirements Model*. International Journal of Web Engineering and Technology (**IJWET**). ISSN: 1476-1289. Vol. 3, No. 1. Pp 4-42. 2007.

Citations: -

International Conferences:

- **Pedro Valderas**, Vicente Pelechano, Gustavo Rossi, Silvia Gordillo. *From Crosscutting Concerns to Web Systems Models*. The 8th International Conference on Web Information Systems Engineering (**WISE 2007**). Nancy, France. December 2007. SPRINGER. Accepted, pending publication.

Citations: -

- **Pedro Valderas**, Vicente Pelechano, Oscar Pastor. *Introducing Graphic Designers in a Web Development Process*. 19th Conference on Advanced Information Systems Engineering (**CAiSE 2007**). Trondheim, Norway. June 2007. SPRINGER. ISBN: 978-3-540-72987-7 ISSN: 0302-9743 (Print) 1611-3349. LNCS 4495. Pp 395-408.

Citations: 1

8. Conclusions

- **Pedro Valderas**, Vicente Pelechano. *Improving Communication in Requirements Engineering Activities for Web Applications*. VII International Conference on Web Engineering (**ICWE 2007**). Como, Italia. July 2007. SPRINGER. ISBN: 978-3-540-73596-0 ISSN: 0302-9743 (Print) 1611-3349. LNCS 4607. Pp 242-247.

Citations: -

- **Pedro Valderas**, Vicente Pelechano, Oscar Pastor. *Towards an End-User Development Approach for Web Engineering Methods*. 18th Conference on Advanced Information Systems Engineering (**CAiSE 2006**). Grand-Duchy of Luxembourg, Luxembourg. June 2006. SPRINGER. LNCS 234. 528-543.

Citations: 2

- Javier Muñoz, **Pedro Valderas**, Vicente Pelechano, Oscar Pastor. *Requirements Engineering for Pervasive Systems. A Transformational Approach*. Poster. 14th IEEE International Requirements Engineering Conference (**RE 2006**). Minneapolis, USA. 9-2006.

Citations: 2

- Gonzalo Rojas, **Pedro Valderas** and Vicente Pelechano. *Describing Adaptive Navigation Requirements Of Web Applications*. IV International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (**AH2006**). Dublin, Ireland, June 20-23, 2006. SPRINGER. ISBN: 3-540-34696. LNCS 4018. Pp 318-322.

Citations: 1

- **P. Valderas**, J. Fons, V. Pelechano. *Developing E-Commerce Applications from Task-Based Descriptions*. 6th Electronic Commerce and Web Technologies (**ECWEB 05**). 23-26 August, Copenhagen, Denmark, 2005. SPRINGER. ISBN: 3-540-28467-2 ISSN: 0302-9743. LNCS 3590. Pp 506-512

Citations: 7

8. Conclusions

- **P. Valderas**, J. Fons, V. Pelechano. *Transforming Web Requirements into Navigational Models: An MDA based Approach*. 24th International Conference on Conceptual Modelling (**ER 2005**). Klagenfurt, (Austria). SPRINGER. ISBN: 3-540-28467-2 ISSN: 0302-9743. LNCS 3716. pp 63-75.

Citations: 7

- **P. Valderas**, J. Fons, V. Pelechano. From Web Requirements to Navigational Design. A Transformational Approach. 5th International Conference on Web Engineering (**ICWE 2005**). 27-29 July, Sydney, Australia, Editors: David Lowe, Martin Gaedke. SPRINGER. ISSN: 0302-9743 ISBN: 3-540-27996-2. LNCS 3579. Pp 506-512

Citations: 3

International Workshops:

- **P. Valderas**, J. Fons, V. Pelechano. Using Task Descriptions for the Specification of Web Application Requirements. VIII International Workshop on Requirements Engineering (**WER 2005**). 13-14 June 2005, Porto, Portugal. ISBN: 972-752-079-0. pp 257-268.

Citations: -

Book Chapters:

- **Pedro Valderas**, Marta Ruiz, Victoria Torres, Vicente Pelechano. *Especificación de Requisitos en el Desarrollo de Aplicaciones Web*. Tendencias en el desarrollo de aplicaciones web. Ed.: Departamento de Informática y Automática de la Universidad de Salamanca. ISBN 84-688-7872-3. Pp 101-118. 2004

Citations: -

Table 7.1 summarizes and classifies these publications according to the place of publication.

8. Conclusions

Table 7.1 Summary of Publications

Category	Number	Acronyms	Citations
International Conference published by SPRINGER in the LNCS serie	8	WISE, CAISE (2), ICWE (2), EC-WEB, ER, AH	23
International Conference published by IEEE	1	RE	-
International Workshop	1	WER	-
International Journal indexed by JCR	1	Information Research	-
International Journal	1	IJWET	-
Book Chapter	1	Tendencias en el desarrollo de aplicaciones web	-
Total	13		23

8.4 A Final Reflection

As a final conclusion, we would like to paraphrase the spiritual teacher and founder of Buddhism, the Siddhārtha Gautama Buddha:

“I never see what has been done; I only see what remains to be done.”

References

A

[AGG] Attributed Graph Grammar System v6.0. <http://tfs.cs.tu-berlin.de/agg/> (last time accessed 22-11-2007).

[Akehurst et al 2003] Akehurst, D., Kent S., and Patrascoiu O. *A Relational Approach to Defining and Implementing Transformations in Metamodels*. In *Software and Systems Modeling* 2(4), 2003, pp. 215–239. Available on-line: <http://www.cs.kent.ac.uk/pubs/2003/1764/> (last time accessed 22-11-2007).

[Al-Rawas et al. 1996] Al-Rawas, A. and Easterbrook, S. *Communication problems in requirements engineering: a field study*. Proc. of the First Westminster Conference on Professional Awareness in Software Engineering, London. 1996.

[Annett & Duncan 1967] Annett, J. and Duncan, K. D. *Task analysis and training design*. *Occupational Psychology*, 41, pp 211- 221. 1967.

B

[Baresi et al. 2001] Baresi, L., Garzotto, F. and Paolini, P. *Extending UML for Modeling Web Applications*. Proceedings of the 34th Hawaii International Conference on System Sciences. 2001.

[Berners-Lee et al. 1994] Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., and Secret, A. *The World-Wide Web*. *Communications of the ACM* Vol 37, No 8, pp. 76-82. Aug. 1994.

[Bézivin et al. 2003] Bézivin, J., Dupé, G., Jouault, F. and Rougui, J. E. *First experiments with the ATL model transformation language: Transforming XSLT into XQuery*. In the online proceedings of the OOPSLA'03 Workshop on Generative Techniques in the Context of the MDA. 2003. Available in: <Http://www.softmetaware.com/oopsla2003/mda-workshop.html> (last time accessed 22-11-2007).

References

- [Boehm 1981] Boehm, B. *Software Engineering Economics*. Englewood Cliffs, Prentice-Hall. 1991.
- [Boehm 1984] Boehm, B. *Model and metrics for software management and engineering*, IEEE Computer Society Press, pp. 4-9. 1984.
- [Boehm 1987] Boehm, B. *Industrial software metrics top 10 list*, IEEE Software, Vol 4, No 5, pp. 84-85. 1987.
- [Boocock 2003] Boocock, P., The Jamda Project, 6 May 2003. Available online: <http://jamda.sourceforge.net/> (last time accessed 22-11-2007).
- [Botella et al. 2001] Botella, P., Burgués, X., Franch, X., Huerta, M., Salazar, G. *Modeling Non-Functional Requirements. Applying Requirements Engineering*, eds. Amador Durán and Miguel Toro (selected papers from JIRA'01). Catedral publicaciones. 2001.
- [Brambilla et al. 2006] Brambilla, M., Ceri, S. and Fraternali, P. *Process Modeling in Web Applications*. ACM Transactions on Software Engineering and Methodology (TOSEM). 2006.
- [Brooks 1995] Brooks, F. P. Jr. *The Mythical Man-Month: Essays on Software Engineering Anniversary Edition*. Addison-Wesley, 1995.
- [Burdman 1999] Burdman, J. *Collaborative Web Development*. Addison-Wesley, Reading, MA. 1999

C

- [Cachero 2003] Cachero, C. *Una extensión a los métodos OO para el modelado y generación automática de interfaces hipermediales*. PhD Thesis (in Spanish). Universidad de Alicante. Alicante, Spain. January 2003.
- [Cachero & Koch 2002a] Cachero, C. and Koch, N. *Conceptual Navigation Analysis: a Device and Platform Independent Navigation Specification*. 2nd International Workshop on Web-oriented Software Technology (IWWOST'02). Málaga, España. Junio 2002.
- [Cachero and Koch 2002b] Cachero, C. and Koch, N. *Navigation Analysis and Navigation Design in OO-H and UWE*. Technical Report. Universidad de Alicante, Spain. April 2002.
- [Card et al. 1983] Card S.K., Moran T.P. and Newell A. *The Psychology of Human Computer Interaction*. Lawrence Erlbaum Associates. 1983.

References

- [Cambridge 2007] Cambridge Dictionaries Online. <http://dictionary.cambridge.org/> (last time accessed 22-11-2007).
- [Ceri et al. 2000] Ceri, S., Fraternali, P. and Bongio, A. *Web Modeling Language (WebML): a Modeling Language for Designing Web Sites*. WWW9 Conference, Amsterdam, May 2000.
- [Ceri et al 2001] Ceri S., Fraternali, P. and Maristella, M. S. *WebML Application Frameworks: a Conceptual Tool for Enhancing Design Reuse*. WWW10 Workshop Web Engineering, Hong Kong, May 2001.
- [Chen 1976] Chen, P.P. *The Entity-Relationship Model: Towards a Unified View of Data*. ACM Transactions on Database Systems, Vol 1 no 1, pp 471-522. 1976.
- [Chung et al. 1999] Chung, L, Nixon, B.A., Yu, E. and Mylopuolos, J. *Non-Functional Requirements in Software Engineering*. International Series in Software Engineering, Vol 5. Sprimger. 1999.
- [Christel et al. 1992] Christel K. and Kang C. *Issues in Requirements Elicitation*. Technical Report CMU/SEI-92-TR-12, Software Engineering Institute, Carnegie Mellon University. 1992.
- [Conallen 1999] Conallen, J. *Building Web Applications with UML*. Addison Wesley. 1999.
- [Conklin 1987] Conklin, J. *Hypertext: An Introduction and Survey*. IEEE Computer, Vol. 20, No 9, pp. 17-41. Sept., 1987.
- [Constantine and Lockwood 1999] Constantine, L.L. and Lockwood, L.A.D. *Software for Use: A practical Guide to the Models and Methods of Usage-Centered Design*. Addison Wesley. 1999.
- [Corradini et al. 1997] Corradini, A., Ehrig H., Heckel R., Korff M., Löwe M., Ribeiro L. and Wagner A. *Algebraic approaches to graph transformation - part I: Single pushout approach and comparison with double pushout approach*. In Rozenberg G. (Ed.), Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations, World Scientific, pp.247-312. 1997.
- [Cowan & Lucena 1995] Cowan, D. D. and Lucena, C. J. P. *Abstract Data Views, An Interface Specification Concept to Enhance Design for Reuse*. IEEE Transactions on Software Engineering, Vol.21, No.3, March 1995.
- [Cunin et al. 2001] Cunin, PY., Greenwood, RM., Francou, L., Robertson, I. and Warboys, B. *The PIE Methodology-Concept and Application*. In Software Process

References

Technology: 8th European Workshop, EWSPT 2001, Witten, Germany, June 19-21, 2001.

[Cutter 2000] Cutter Consortium. *Poor Project Management Number-one Problem of Outsourced E-projects*. CutterResearch Briefs, November. <http://www.cutter.com/research/2000/crb001107.html> (last time accessed 22-11-2007). November, 2000.

[Cutter IT Journal 2001] *Cutter IT Journal*, Vol 14, No. 7, July 2001

[Cysneiros et al. 2001] Cysneiros L.M., Leite, J. and Neto, J. *A Framework for integrating non-functional requirements into conceptual models*. Requirements Engineering, Springer London. Vol. 6, N° 2, pp 97-155. 2001.

[Czarnecki & Helsen 2003] Czarnecki, K. and Helsen, S. *Classification of Model Transformation Approaches*. In Online Proceedings of the OOPLSLA'03 workshop on Generative Techniques in the Context of Model Driven Architectures, 2003.

D

[Davis 1993] Davis, A. M. *Software Requirements: Objects, Functions and States*. Prentice-Hall, 2nd ed., 1993.

[De Lara & Vangheluwe 2002] De Lara, J. and Vangheluwe, H. *AToM3: A Tool for Multi-formalism Modelling and Meta-modelling*. LNCS 2306, pp.:174-188. 2002. See: <http://atom3.cs.mcgill.ca> (last time accessed 22-11-2007).

[De Lara et al. 2005] De Lara, J., Bardohl, R., Ehrig, H., Ehrig, K. and Prange, U. *Gabriele Taentzer3Attributed Graph Transformation with Node Type Inheritance*. Theoretical Computer Science (Elsevier), 376(3): 139-163. Special Section on FASE'04 and FASE'05. 2007.

[De Troyer & Leune 1998] De Troyer, O. and Leune, C. *WSDM: A User-Centered Design Method for Web Sites*. Computer Networks and ISDN systems, Proceedings of the 7th International World Wide Web Conference, pp. 85 - 94, Publ. Elsevier, Brisbane, Australia. 1998.

[DeMarco 1979] DeMarco T. *Structured analysis and system specification*. Yourdon Press. ISBN:0-917072-14-6. 1979

[Depke et al. 2002] Depke, R., Heckel, R. and Küster, J.M., Formal Agent-Oriented Modeling with UML and Graph Transformation, in Science of Computer Programming, 44(2), pp. 229-252. August 2002.

References

[Deshpande & Hansen 2001] Deshpande, Y. and Hansen, S. *Web Engineering: Creating a Discipline among Disciplines*, IEEE Multimedia, Special issues on Web Engineering, Vol 8, No 2, pp 82-87. 2001.

[Diaz et al. 1995] Diaz A., Isakowitz T., Maiorana V. and Gilabert G. *RMC: A tool to design WWW applications*. In Proceedings of the 5th International World Wide Web Conference. 1995.

[Didonet Del Fabro et al. 2006] Didonet Del Fabro, M, Bézivin, J, and Valduriez, P. *Weaving Models with the Eclipse AMW plugin*. In: Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany, 2006.

[Didonet Del Fabro et al. 2005] Didonet Del Fabro M, Bézivin J, Jouault F, Breton E. and Gueutas G. *AMW: a generic model weaver*. In: Proc. of 1ère Journée sur l'Ingénierie Dirigée par les Modèles, Paris, France. pp 105-114. 2005.

[Dijkstra 1972] Dijkstra, E.W. *The Humble Programmer*, in Communication of the ACM, Vol 15, No 10, pp. 859-866. 1972.

[Diaper 1990] Diaper, D. *Task Analysis for Human-Computer Interaction*. Prentice Hall PTR Upper Saddle River, NJ, USA. 1990.

[DMOF 2002] dMOF 1.1, An OMG Meta Object Facility Implementation, The Corba Service Product Manager, University of Queensland. 2002.

[Durán et al. 1999] Durán, A., Bernárdez, B., Ruiz A. and M. Toro. *A Requirements Elicitation Approach Based on Templates and Patterns*. International Workshop on Requirements Engineering (WER'99). Buenos Aires (Argentina), pp. 17-29. 1999.

[Durán 2000] Durán, A. *A Methodological Framework for Requirements Engineering of Information Systems*. PhD thesis (in Spanish), University of Seville, 2000.

E

[Eclipse] Eclipse Open Development Platform. [Http://www.eclipse.org/](http://www.eclipse.org/) (last time accessed 22-11-2007).

[Ehrig 1979] Ehrig H. *Introduction to the algebraic theory of graph grammars - a survey* In Proceedings International Workshop on Graph Grammars and their Application to Computer Science and Biology, Springer-Verlag, pp. 1-69. 1979.

References

[Ehrig et al. 1985] Ehrig, H. and Mahr, B. *Fundamentals of Algebraic Specifications 1: equations and Initial Semantics*. Vol. 6 of EATCS Monographs on Theoretical Computer Science. Springer, 1985.

[Ehrig & Habel 1986] Ehrig H. and Habel A. *Graph grammars with application conditions*. in Rozenberg G., and Salomaa A. (Eds.), *The Book of L*, Springer-Verlag, pp. 87-100. 1986.

[Ehrig et al. 2004] Ehrig, H., Taentzer, G. and Prange, U. *Fundamental theory for typed attributed graph transformation*. Springer. 2004.

[EJB] Enterprise JavaBeans Technology. Sun Microsystems. <http://java.sun.com/products/ejb/> (last time accessed 22-11-2007).

[England & Finney 1999] England, E. and Finney, A. *Managing multimedia: project management for interactive media*. Second edition. Addison-Wesley, Reading, MA. 1999.

[Escalona et al. 2003] Escalona, M.J., Torres, J., Mejías, M. and Reina, A. *NDT-Tool: A CASE Tool to deal with Requirements in Web Information Systems*. In Proc. International Conference on Web Engineering (ICWE). LNCS 2722, pp.212–213. Berlin Heidelberg. 2003.

[Escalona 2004] Escalona M.J. *Modelos y técnicas para la especificación y el análisis de la navegación en sistemas software*. PhD. Thesis (in Spanish). Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. 2004.

[Escalona & Koch 2004] Escalona M.J., and Koch, N. *Requirements engineering for web applications: A comparative study*. Journal on Web Engineering, Rinton Press, 2(3), 193-212. 2004.

F

[Fons et al. 2002] Fons, J., Valderas, P. and Pastor, O. *Specialization in Navigational Models*. In Argentine Conference on Computer Science and Operational Research. Subserie ICWE, Iberoamerican Conference on Web Engineering. Volume 31 of Anales JAIO. 2002.

[Fons et al 2003] Fons, J., Pelechano, V., Albert, M. and Pastor O. *Development of Web Applications from Web Enhanced Conceptual Schemas*. 22th International Conference on Conceptual Modeling (ER 2003), volume 2813 of LNCS. 2003

[Fowler 1997] Fowler, M. *Analysis Patterns: Reusable Object Models*. Addison-Wesley. 1997.

References

[Fraternali et al. 2006] Fraternali, P., Ceri, S., Tisi, M. and Tosetti, E. *Developing eBusiness solutions with a Model Driven Approach*. IMP2006. <http://www.webml.org/webml/upload/ent5/1/IMP2006FraternaliFull.pdf> (last time accessed 22-11-2007).

G

[Gallian 1998] Gallian, J.A. *A dynamic survey of graph labeling*. Electronic Journal of Combinatorics, 1998

[GAO 1979] *Contracting for Computer Software Development: Serious Problems Require Management Attention to Avoid Wasting Additional Millions*. Report FGMSD-80-4, U. S. Government Account Office, Noviembre 1979. Although to find this document may be a difficult task, it is commented and referenced in [Durán 2000], [Davis 1993], [Christel et. al 1992] and [Goguen 1994].

[Garrigos et al. 2005] Garrigos, I., Gomez, J., Barna, P. and Houben, G.J. 2005. A Reusable Personalization Model in Web Application Design. In 2nd Workshop on Web Information Systems Modelling (WISM 2005), in conjunction with ICWE 2005, Sydney, Australia. 2005.

[Garzotto et al. 1993] Garzotto, F., Paolini, P. and Schwabe, D. *HDM - A Model-Based Approach to Hypertext Application Design*. ACM Transactions on Information Systems, Vol. 11, IWO. 1, Pages 1-26. January 1993.

[Gellersen & Gaedke 1999] Gellersen, H.W. and Gaedke, M. *Object-oriented Web application development*. Internet Computing, IEEE, Vol 3, No 1, pp 60-68. 1999.

[Gerber et al. 2002] Gerber, A., Lawley, M., Raymond, K., Steel, J. and Wood, A. *Transformation: The Missing Link of MDA*. In Proceedings of the 1st International Conference on Graph Transformation ICGT'02. LNCS, Vol. 2505. Springer-Verlag, pp. 90-105. October 2002.

[Glass 2001] Glass, R. *Who's Right in the Web Development Debate?*, Cutter IT Journal, Vol 14, No. 7, pp 6-10. 2001.

[Ginige 2001] Ginige A., Murugesan S. Guest Editors' Introduction: *Web Engineering: An Introduction*. IEEE MultiMedia Vol 8, No 1, pp14-18. 2001

[Goguen 1994]. Goguen J. A. *Requirements Engineering as the Reconciliation of Social and Technical Issues*. Requirements Engineering: Social and Technical Issues, pp. 165-199. Academic Press. 1994.

References

[Gomez et al. 2000] Gomez, J., Cachero, C. and Pastor, O. *Extending a Conceptual Modelling Approach to Web Application Design*. In Proc. of the 12th International Conference on Advanced Information Systems Engineering (CAISE). LNCS 1789. pp 79–93. 2000.

[Gotel 1995] Gotel, O. *Contribution Structures for Requirements Traceability*. Ph.D. thesis. Imperial College. Department of Computing. London-England. 1995.

[Grechenig & Tscheligi 1993] Grechenig, T. and Tscheligi, M. *Human Computer Interaction: Vienna Conference, VHCI '93, Fin de Siècle*, Vienna, Austria. Springer. 1993.

[Greenspun 1999] Greenspun, P. *Philip and Alex guide to Web publishing*. <http://photo.net/wtr/thebook>. 1999 (last time accessed 22-11-2007).

H

[Habel et al. 1996] Habel A., Heckel R. and Taentzer G. *Graph grammars with negative application conditions*. in Fundamenta Informaticae, 26(3), 1996.

[Heckel & Wagner 1995] Heckel R. and Wagner A. *Ensuring consistency of conditional graph grammars - a constructive approach*. In Lecture Notes in Theoretical Computer Science, Springer Verlag, 1995.

[Heckel et al. 2002] Heckel, R., Mens, T. and Wermelinger, M. (eds.), *Proceedings of the Workshop on Software Evolution through Transformations: Toward Uniform Support throughout the Software Life-Cycle*. Electronic Notes in Theoretical Computer Science 72(4). 2002.

[Heckel 2004] R. Heckel. *Graph transformation in a nutshell*. In Proceedings of the School on Foundations of Visual Modelling Techniques (FoVMT) of the SegraVis Research Training Network. 2004.

[Hofman 2000] Hofman, H.F. *Requirements Engineering: A Situated Discovery Process*. Gabler, Wiesbaden, Germany. 2000.

[Holzschlag 2001] Holzschlag, M.E. *How Specialization Limited the Web*, WebTechniques, <http://www.webtechniques.com/archives/2001/09/desi/> (last time accessed 22-11-2007). Sept 2001.

[Ho et al. 1999] Ho, W.M., Jézéquel, J.M., Le Guennec, A. and Pennaneach, F. *UMLAUT: An Extensible UML Transformation Framework*. In Proceedings of the 14th IEEE International Conference on Automated Software Engineering ASE'99 pp. 275–279, IEEE Computer Society Press, Los Alamitos, 1999.

References

I

[IEEE 1998] IEEE, Guide to Software Requirements Specifications. 1998: ANSI/IEEE Standard 830-1998.

[Insfrán et al. 2002] Insfrán, E., Pastor, O. and Wieringa, R. Requirements Engineering- Based Conceptual Modelling. Requirements Engineering Journal, Vol 7 (1). 2002.

[Insfrán 2003] Insfrán, E. *A Requirements Engineering Approach for Object-Oriented Conceptual Modeling*. PhD Thesis. Department of Information Systems and Computation. Technical University of Valencia. Spain. October 2003.

[Isakowitz et al. 1995] Isakowitz T., Stohr E. and Balasubramanian P. *A methodology for the design of structured hypermedia applications*. Communications of the ACM, 8(38), 34-44. 1995.

[ISO 1995] International Standards Organization. Information Technology: Software Life-cycle Processes (ISO/IEC12207). Geneva, Switzerland, 1995.

J

[Jaaksi 1998] Jaaksi, A. *Our CASEs with Use CASEs*. Journal of Object-Oriented Programming (JOOP). Vol 10, N°9, pp 58-65. February 1998.

[Jacobson & Jacobson 1995] Jacobson, I. and Jacobson, S. *Reengineering your software engineering process*. Object Magazine, March 1995.

[Jacobson et al. 1999] Jacobson, I., Booch, G. and Rumbaugh, J. *The Unified Software Development Process*. Addison-Wesley Professional. 1999.

[Jeenicke et al. 2003] Jeenicke, M., Bleek, W.G. and Klischewski, R. *Revealing Web User Requirements through e-Prototyping*. In Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE 03), San Francisco, USA, July 1 - 3, 2003.

[John and Kieras 1996] John, B. E. and Kieras, D. E. *Using GOMS for user interface design and evaluation: Which technique?* ACM Transactions on Computer-Human Interaction, 3, 287-319. 1996.

[Johnson 1999] Johnson P. *Tasks and situations: considerations for models and design principles in human computer interaction*. In Proc. of HCI International, pp.1199–1204. 1999.

References

[Johnson et al. 1992] Johnson, P., Markopoulos, M. and Johnson, H. *Task Knowledge Structures: A Specification of user task models and interaction dialogues*. Proceedings of Interdisciplinary Workshop on Informatics and Psychology, Austria. 1992.

K

[Kappel et al. 2001] Kappel, G., Pröll, B., Retschitzegger W. and Schwinger, W. *Modelling Customizable Web Applications - A requirement's Perspective*. International Workshop on Data Semantic in Web Information Systems. Kyoto, Japan 2001.

[Kreowski et al. 2006] Kreowski, H.J., Klempien-Hinrichs, R. and Kuske, S. *Some Essentials of Graph Transformation*. In Recent Advances in Formal Languages and Applications, volume 25 of Studies in Computational Intelligence, pp 229-254. Springer, 2006

[Kirwan & Ainsworth 1992] Kirwan, B. and Ainsworth, L (eds). *K. A Guide to Task Analysis*. Taylor & Francis Ltd. 1992.

[Kobsa 2001] Kobsa, A. *Generic User Modeling Systems*. User Modelling and User-Adapted Interaction. Vol. 11 (1-2), Ten Year Anniversary Issue, , pp. 49-63. 2001.

[Koch 2000] Koch, N. *Software Engineering for Adaptive Hypermedia Applications*. PhD thesis, Ludwig-Maximilians-University, Munich, Germany, 2000.

[Koch et al. 2006] Koch, N., Zhang, G. and Escalona, M.J. *Model Transformations from Requirements to Web System Design*. ICWE'06, July 11-14, 2006, Palo Alto, California, USA. 2006.

L

[Lange 1996] Lange D. *An object-oriented design approach for developing hypermedia information systems*. Journal of Organizational Computing and Electronic Commerce, 6(3),269-293. 1996.

[Lauesen 2003] Lauesen, S. *Task Description as Functional Requirements*. IEEE Software March-April 2003, Volume 20, Issue 2, Pages 58-65. 2003.

[Lee et al. 1998] Lee, H., Lee, C. and Yoo, C. 1998. *A Scenario-Based Object-Oriented Methodology for Developing Hypermedia Information Systems*. In Proceedings of the Thirty-First Annual Hawaii international Conference on System Sciences-Volume 2. January 06 - 09, 1998.

References

- [Leite et al. 1997] Leite, J.C., Rossi, G. , Balaguer, F., Maiorana, V., Kaplan, G., Hadad, G., and Oliveros, A. *Enhancing a Requirements Baseline with Scenarios*. Third IEEE International Symposium on Requirements Engineering (RE'97). Annapolis, Maryland, USA. pp 44-53. 1997.
- [Limbourg 2004] Limbourg Q. *Multi-Path Development of User Interfaces*. PhD. Thesis. Université catholique de Louvain. 2004.
- [Liu & Yu 2001] Liu, L. and Yu. E. *From Requirements to Architectural Design using Goals and Scenarios*. Proceedings of the 6th Micon Workshop. Canada. 2001.
- [Livesey & Guinane 1997] Livesey D. and Guinane T. *Developing Object-Oriented Software, An Experience-Based Approach (IBM's OOTC)*. Prentice Hall. 1997
- [Lowe & Hall 1999] Lowe, D. and Hall, W. *Hypermedia and the Web. An Engineering approach*. John Wiley & Son. 1999.
- [Lowe & Elklund 2002] Lowe D. and Elklund, J. *Client Needs and the Design Process in Web Projects*. Web Engineering Track of the WWW2002 Conference. 2002.
- [Lowe 2003] Lowe, D. *Web system requirements: an overview*. Requirements Engineering Vol. 8 pp 102–113. Springer-Verlag London Limited. 2003.
- [Löwe 1993] Löwe M. *Algebraic approach to single-pushout graph transformation*. In Theoretical Computer Science, Vol. 1. pp. 181-224. 1993.

M

- [Macdonald & Welland 2001] McDonald, A. and Welland, R. *Web Engineering in Practice*. Proceedings of the 4th Workshop on Web Engineering (in conjunction with 10th Int. Conf. on WWW), Hong Kong. 2001.
- [Mandel et al. 1998] Mandel, L., Koch, N. and Maier, C. *Extending UML to Model Hypermedia and Distributed Systems*. Technical Report 9804, Ludwig-Maximilians-Universität München, Institut für Informatik, 1998. Online available at: <http://www.fast.de/Projekte/forsoft/intoohdm/index.html> (last time accessed 22-11-2007).
- [March & Smith 1995] March S and Smith G. *Design and Natural Science Research on Information Technology*. Decision Support Systems 15, 251 - 266. DOI: 10.1016/0167-9236(94)00041-2. 1995.

References

- [Matera et al. 2003] Matera, M., Maurino, S. Ceri, S. and Fraternali, P. *Model-Driven Design of Collaborative Web Applications*. Software-Practice & Experience, Vol 33, pp 1-32. 2003.
- [MDA] Model Driven Architecture. Object Management Group. <http://www.omg.org/mda/> (last time accessed 22-11-2007).
- [Mecca et al. 1999] Mecca, G., Atzeni, P. and Crescenzi, V. The ARANEUS Guide to Web-Site Development. Technical Report, University of Rome, Italy. 1999.
- [Mellor et al. 2003] Mellor, S.J., Clark, A.N. and Futagami, T. *Model-driven development - Guest editor's introduction*. IEEE Software, Vol. 20 N° 5, pp 14- 18, Sept.-Oct. 2003.
- [Mendes et al. 2006] Mendes, E. and Mosley, N. *Web Engineering: Theory and Practice of Metrics and Measurement for Web Development*. Springer-Verlag, ISBN: 3-540-28196-7. 2006
- [Mens et al. 2001] Mens, T., Van Eetvelde, N., Janssens, D. and Demeyer, S., *Formalising Refactoring with Graph Transformations*. In Fundamenta Informaticae, 21, pp. 1001–1022. 2001.
- [Miller 1956] Miller, G. *The magical number seven plus or minus two: some limits on our capacity for processing information*. Psychological Review 63, pp. 81–97. 1956.
- [MOF] Meta Objects Facilitates (MOF) 2.0. Object Management Group. <http://www.omg.org/> (last time accessed 22-11-2007).
- [Montanari 1970] Montanari, U.G., *Separable Graphs, planar Graphs and Web Grammars*. In Inf. Contr., 16, pp. 243-267. 1970.
- [Murugesan et al. 1999] Murugesan, S., Deshpande, Y., Hansen, S. and Ginige, A. *Web Engineering: A New Discipline for Development of Web-based Systems*, Proceedings of the First ICSE Workshop on Web Engineering, International Conference on Software Engineering. 1999.

N

- [Nakajo & Kume 1991] Nakajo, T. and Kume, H. *A CASE History Analysis of Software Error Cause-Effect Relationships*. Transactions on Software Engineering, 17(8): 830-838. 1991.

References

[Nanard & Nanard 1995] Nanard J., and Nanard M. *Hypertext design environments and the hypertext design process*. Communication of the ACM, Vol 38 (8), 49-56. August 1995.

O

[OCL] Object Constraint Language (OCL) 2.0. Object Management Group. [Http://www.omg.org](http://www.omg.org) (last time accessed 22-11-2007).

[Olivanova] Olivanova: The programming Machine. Care Technology. Information available at <http://www.care-t.com> (last time accessed 22-11-2007).

[Olsina 1998] Olsina, L. *Building a Web-based information system applying the hypermedia flexible process modeling strategy*. 1st International workshop on Hypermedia Development, Hypertext 1998.

[Omerod et al. 2003] Ormerod, T.C. and Shepherd, A. *Using task analysis for information requirements specification: The SGT method*. Chapter 16 in D. Diaper & N. Stanton (Eds.) (in press) *The Handbook of Task Analysis for Human-Computer Interaction*. London: Lawrence Erlbaum Associates. 2003

[OptimalJ] Compuware OptimalJ: Model-driven development for Java. <http://www.compuware.com/products/optimalj/> (last time accessed 22-11-2007).

[Overmyer 2000] Overmyer, S.P. *What's different about requirements engineering for web sites?* Requirements Engineering Vol. 5 pp 62–65. Springer-Verlag London Limited. 2000.

P

[Pan et al. 2001] Pan, D., Zhu, D. and Johnson, K. *Requirements Engineering Techniques*. Internal Report. Department of Computer Science. University of Calgary. Canada. 2001.

[Partsch & Steinbruggen 1983] Partsch, H. and Steinbruggen, R. *Program Transformation Systems*. ACM Computing Surveys 15, 3, pp 199–236. September 1983.

[Pastor et al. 1997] Pastor, O., Insfran, E., Pelechano, V., Romero, J. and Merseguer, J. *OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods*. In Proc. of the 9th International Conference on Advanced Information Systems Engineering (CAISE). LNCS 1250. pp 145–158. 1997.

References

[Pastor et al. 2001] Pastor O., Gómez J., Insfrán E., and Pelechano V., The OO-Method approach for Information Systems Modelling: from Object-Oriented Conceptual Modeling to Automated Programming. *Information Systems*, 26(7): p. 507-534. 2001.

[Pastor et al. 2005] Pastor, O., Fons, J., Pelechano, V., and Abrahão, S. *Conceptual modelling of Web applications: the OOWS approach*. Book chapter in *Web Engineering - Theory and Practice of Metrics and Measurement for Web Development*, Mendes E. (Eds.), Springer 2005.

[Paternò 1997] Paternò F., Mancini C., Meniconi S. *ConcurTaskTree: a Diagrammatic Notation for Specifying Task Models*. *INTERACT'97*, Chapman & Hall, 362-369. 1997.

[Pfalz & Rozenberg 1969] Pfalz, J. L. and Rozenberg, A., *Web Grammars*. Proceedings of the International joint Conference on Artificial intelligence, Washington, pp. 609-619. 1969.

[Pressman 2001] Pressman, R.S. *Web Engineering: An Adult's Guide to Developing Internet-Based Applications*, Cutter IT Journal, Vol 14, No. 7, pp 2-5. 2001.

[Pressman 2004] Pressman, R.S. *Software Engineering: A Practitioner's Approach*. MacGraw-Hill. ISBN: 0-07-285318-2. 2004.

Q

[QVT] Meta Object Facilities (MOF) Query / Views / Transformations (OCL) 1.0. Object Management Group. [Http://www.omg.org](http://www.omg.org) (last time accessed 22-11-2007).

R

[Raghavan et al. 1994] Raghavan, S., Zelesnik, G. and Ford, G. *Lectures Notes of Requirements Elicitation. Educational Materials*. CMU/SEI-94-EM-10. 1994.

[Reifer 2000] Reifer D.J. *Web Development: Estimating Quick-to-Market Software*. IEEE Software. Vol 17, Issue 6, pp: 57–64. Nov.-Dec. 2000.

[Rojas et al. 2006] Rojas, G., Valderas, P. and Pelechano, V. *Describing Adaptive Navigation Requirements Of Web Applications*. IV International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH2006). SPRINGER. LNCS 4018. pp 318-322. Dublin, Ireland, June 20-23, 2006.

[Rosenborg and Scott 1999] Rosenberg D. and Scott K. *Use CASE Driven Object Modelling with UML*. Addison Wesley, 1999.

References

[Rozenberg 1997] Rozenberg, G. (ed.). *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific, Singapore. 1997.

[Rumbaugh et al. 1990] Rumbaugh, J., Blaha, M., Lorensen, W., Eddy, F., and Premerlani, W. *Object-Oriented Modeling and Design*. Prentice Hall; United States Ed. Edition October 1, 1990.

S

[Schwabe & Rossi 1994] Schwabe, D. and Rossi, G. *From Domain Models to Hypermedia Applications: An Object-Oriented Approach*. In International Workshop on Methodologies for Designing and Developing Hypermedia Applications. Edinburgh. 1994.

[Schwabe et al. 1996] Schwabe, D. Rossi, G. and Barbosa S.D.J. *Systematic Hypermedia Application Design with OOHDM*. In Proceedings of the Seventh ACM Conference on Hypertext. Bethesda, Maryland, United States, March 16 - 20, 1996. HYPERTEXT '96. ACM, New York, NY, pp 116-128.

[Segura et al. 2007] Segura, S., Benavides, D., Ruiz-Cortés, A. and Escalona, M.J. *From Requirements to Web System Design. An Automated Approach using Graph Transformations*. IV Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM'07). In conjunction to the XII Jornadas de Ingeniería del Software y Bases de Datos. Zaragoza, Spain. 2007. <http://www.taro.ull.es/dsdm07>. (last time accessed 22-11-2007).

[Sendall & Kozacynski 2003] Sendall, S. and Kozacynski, W. *Model Transformation: The Heart and Soul of Model-Driven Software Development*. In IEEE Software, Vol 20, Issue 5, pp. 42-45. 2003.

[Shepherd 1993] Shepherd, A. *An approach to information requirements specification for process control tasks*. Ergonomics, 36, pp 807 – 819. 1993.

[Shepherd 2001] Shepherd, A. *Hierarchical Task Analysis*. Taylor & Francis, London. 2001.

[Silberschatz et al. 2004] Silberschatz, A., Galvin, P.B. and Greg, G. *Operating System Concepts*. Hoboken, NJ: John Wiley & Sons. ISBN 978-0-471-69466-3. 2004

[Sommerville 1982] Sommerville, I. *Software engineering* (1st ed.), Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1982.

[Sommerville & Sawyer 1997] Sommerville I., Sawyer P. *Requirements Engineering. A good practice guide*. John Wiley and Sons, Inc., 1997.

References

[Souer et al. 2006] Souer, J., Van De Weerd, I., Versendaal, J. and Brinkkemper, S. *Situational requirements engineering for the development of Content Management System-based web applications*. International Journal of Web Engineering and Technology (IJWET), Vol. 3, No. 4, 2007.

[Sawyer & Kotonya 2001] Sawyer P. and Kotonya G. *Software Requirements*. Chapter 2 of the IEEE SWE Book Project Report.

[SPem 2002] Software Process Engineering Metamodel, version 1.1. Object Management Group. <http://www.omg.org/technology/documents/formal/spem.htm>. (last time accessed 22-11-2007). 2002

[Suh 2004] Suh, W. *Web Engineering: Principles and Techniques*. Idea Group Inc (IGI). ISBN: 1591404339. 2004

T

[Thomson et al. 1998] Thomson J., Greer J. and Cooke J. *Algorithmically detectable design patterns for hypermedia collections*. In Proceedings of Workshop on Hypermedia development Process, Methods and Models, Hypertext '98. 1998.

[TRL 2003] Alcatel, Softeam, Thales, TNI-Valiosys, Codagen Corporation et al. MOF Query/Views/Transformations, Revised Submission. OMG Document: ad/03-08-05. Available at <http://www.omg.org/cgi-bin/doc?ad/03-08-05> (last time accessed 22-11-2007).

[TSG 1995] TSG. *The CHAOS Report*. The Standish Group. Available in http://www.standishgroup.com/sample_research/chaos_1994_1.php. (last time accessed 22-11-2007, registration required). 1995.

U

[UML] Unified Modelling Language 2.0. Object Management Group. <Http://www.uml.org> (last time accessed 22-11-2007).

[Universalis] Univers@lis: France Telecom. Available online: <http://universalis.elibel.tm.fr/site/> (last time accessed 22-11-2007).

V

[Vaishnavi & Kuechler 2004] Vaishnavi, V. and Kuechler, W. 2004. *Design Research in Information Systems*.

References

- [Valderas et al. 2005a] Valderas, P., Fons, J. and Pelechano, V. *Developing E-Commerce Applications from Task-Based Descriptions*. 6th Electronic Commerce and Web Technologies (ECWEB 05). 23-26 August, Copenhagen, Denmark, 2005.
- [Valderas et al. 2005b] Valderas, P., Fons, J. and Pelechano V. *Transforming Web Requirements into Navigational Models: An MDA based Approach*. 24th International Conference on Conceptual Modelling (ER 2005). Klagenfurt, (Austria).
- [Valderas 2007a] Valderas, P. *An Amazon like E-Commerce Application: A CASE Study. Task-Based Requirements Specification-OOWS Conceptual Model Derivation-Web Application Prototype Generation*. Technical Report. Department of Information Systems and Computation. Technical University of Valencia, Spain. 2007. Online available at: <https://oomethod.dsic.upv.es/files/TechnicalReport/CASEStudy.pdf> (last time accessed 22-11-2007).
- [Valderas 2007b] Valderas, P. *A Set of Graph Transformation Rules for Transforming Requirements Specifications into OOWS Conceptual Models*. Technical Report. Department of Information Systems and Computation. Technical University of Valencia, Spain. 2007. Online available at: <https://oomethod.dsic.upv.es/files/TechnicalReport/GraphTransformationRuleCatalogue.pdf> (last time accessed 22-11-2007).
- [Vallecillo et al. 2007] Vallecillo, A., Koch, N., Cachero, C., Comai, S., Fraternali, P., Garrigós, I., Gómez, J., Kappel, G., Knapp, A., Matera, M., Meliá, S., Moreno, N., Pröll, B., Reiter, T., Retschitzegger, W., Rivera, J.E., Schauerhuber, A., Schwinger, W., Wimmer M. and Zhang G. *MDWEnet: A Practical Approach to Achieving Interoperability of Model-Driven Web Engineering Methods*. A position paper presented in the MDWEnet workshop in Como, Italy. July 2007.
- [Valverde et al 2007] Valverde, P., Valderas, P., Fons, J. and Pastor, O. *A MDA-based Environment for Web Applications Development: From Conceptual Models to Code*. 6th International Workshop on Web-Oriented Software Technologies. 2007.
- [Varró 2002] Varró, D. *A Formal Semantics of UML Statecharts by Model Transition Systems*. In Proceedings of the 1st International Conference on Graph Transformation ICGT'02. LNCS, Vol. 2505. Springer-Verlag, pp. 378-392. October 2002).
- [Varró & Pataricza 2004] Varró, D. and Pataricza, A.: *Generic and meta-transformations for model transformations engineering*. In 7th International Conference on the Unified Modeling Language (UML 2004).
- [Veer et al. 1996] Veer, G.C. van der, Lenting, B.F. and Bergevoet, B.A.J. *GTA: GroupWare Task Analysis - Modelling Complexity*. Acta Psychologica 91, pp. 297-322, 1996.

References

[Vilain et al. 2000a] Vilain, P., Schwabe, D. and Sieckenius, C. *Use CASEs and Scenarios in the Conceptual Design of Web Application*. Technical Report MCC 12/00. Departamento de Informática. PUC-Rio. Rio de Janeiro, Brasil, 2000.

[Vilain et al. 2000b] Vilain, P., Schwabe, D. and Sieckenius, C. *A diagrammatic Tool for Representing User Interaction in UML*. Lecture Notes in Computer Science. UML'2000. York, England 2002.

[Vilian & Schwabe 2002] Vilain P., Schwabe D. *Improving the Web Application Design Process with UIDs*. 2nd International Workshop on Web Oriented Software Technology (IWWOST'2002) Málaga, Spain June 10, 2002.

[VMTS] Visual Modeling and Transformation System (VMTS), <http://avalon.aut.bme.hu/~tihamer/research/vmts> (last time accessed 22-11-2007).

W

[Westland 2003] Westland, J. *Project Management Guidebook*. ISBN: 0-473-10445-8. Online available at: <http://www.method123.com/free-project-management-book.php>. (last time accessed 22-11-2007).

[Wirfs-Brock et al. 1990] Wirfs-Brock R., Wilkerson B., Wiener L. *Designing Object-Oriented Software*. Prentice-Hall, 1990.

X

[XMI] XML Metadata Interchange (XMI), v2.1. Object Management Group. <http://www.omg.org/technology/documents/formal/xmi.htm> (last time accessed 22-11-2007).

[XSLT] XSL Transformations (XSLT), v1.0. W3C Recommendation 16 November 1999. <http://www.w3.org/TR/xslt> (last time accessed 22-11-2007).

Y

[Yogesh et al. 2002] Yogesh, D., Murugesan, S., Ginige, A., Hansen, S., Schwabe, D., Gaedke, M., White, B. *Web Engineering*. Journal of Web Engineering, Vol. 1, No.1 003-017. 2002.

[Yoo & Bieber 1998] Yoo, J. and Bieber, M. 2001. A Systematic Relationship Analysis for Modeling Information Domains. <http://citeseer.ist.psu.edu/312025.html>. (last time accessed 22-11-2007). 1998.

References

[Young 2003] Young, R. *The Requirements Engineering Handbook*. Norwood, MA, USA: Artech House, Incorporated, 2003.

[Yourdon 1993].Yourdon, E. *Yourdon Systems Method: Model-Driven Systems Development*. Prentice-Hall. 1993.

Z

[Zhu et al. 2002] Zhu, H., Jin, L., Diaper, D., Bai, G. *Software requirements validation via task analysis*. Journal of Systems and Software, Volume 61, Issue 2, pp 145-169. 15 March 2002.

Appendix A

Meta-Model of the Task-based Requirements Model

A1 Introduction

In this Appendix, we introduce the meta-model of the task-based requirements model presented in Chapter 4. This meta-model is defined according to the Meta Object Facilities (MOF) [MOF]. It is organized around three packages that contain the different elements that define the model. Figure A1 shows these packages.

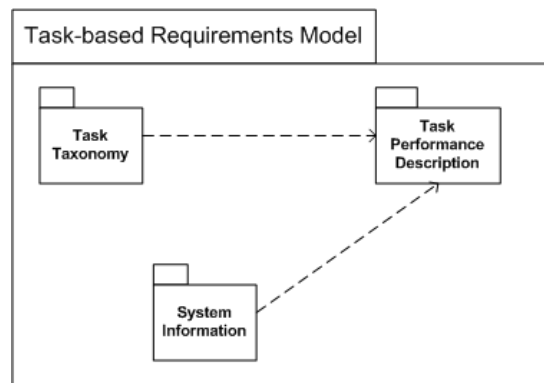


Figure A1 Meta-model packages

The **Task Taxonomy** package defines the elements and constraints that are needed to create a task taxonomy. These elements are related to elements of the **Task Performance Description** package. This package defines the elements and

Appendix A

constraints that allow us to create the task performance description of elementary tasks. Finally, the **System Data** package defines the elements and constraint that must be used to specify the informational requirements of a Web application. These elements are associated to elements of the package Task Performance Description.

In order to specify both the elements of a package and the relationships between them, we use a class diagram. In order to express the different constraints that are associated to the elements of each package we use OCL (Object Constrain Language) [OCL]. Next subsections show the definition of each package.

A2 Task Taxonomy Package

Figure A2 shows the class diagram that defines both the elements and the relationships between elements of the Task Taxonomy package.

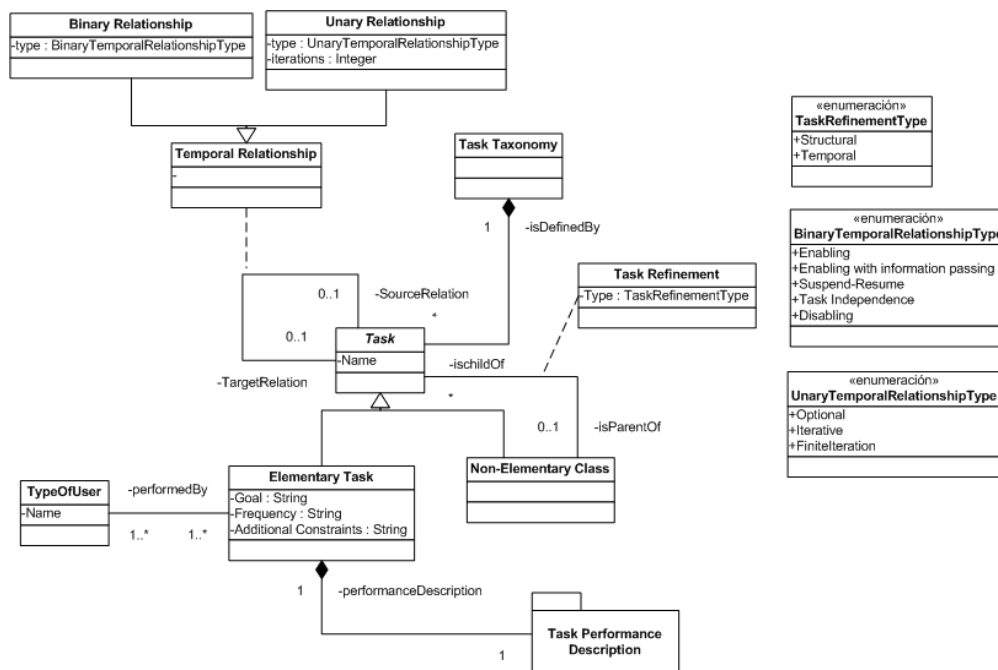


Figure A2 Class diagram of the Task Taxonomy package.

The class diagram shown in Figure A2 defines the Task Taxonomy package as follows:

- The main element in this package is the *Task*. Tasks have a *Name*. There are two types of tasks: *Elementary Task* and *Non-Elementary Task*.

Appendix A

- Non-elementary tasks may be refined into other tasks (elementary or not). The refinement is performed by a *Task Refinement*. There are two types of Task Refinements (see the enumeration *TaskRefinementType*): *Structural* or *Temporal*. A task can be the result of the refinement of an only one task (that is, a task can only have one parent task in the task taxonomy).
- Elementary Tasks present several attributes (*Goal*, *Frequency* and *Additional Constraints*) that characterize them. Furthermore they are related to a set of *TypeOfUsers*. This relationship indicates the types of user that can perform the elementary tasks. Finally, elementary tasks are associated to an element of the Task Performance Description package which indicates how the elementary task must be performed.
- Tasks, moreover, may be related to other tasks by means of a *Temporal Relationship*. There are two types of temporal relationships: *Unary Relationship* and *Binary Relationship*. A task can only be related to an only one task.
- Unary Relationships present the attribute *Type* which indicates its type. This type can be (see the enumeration *UnaryTemporalRelationshipType*): *Optional*, *Iterative* or *Finite Iteration*. Additionally, the attribute *Iterations* indicate the number of times that the task must be performed if the temporal relationship Finite Iteration is used.
- Binary Relationships also present the attribute *Type* which indicates its type. This type can be (see the enumeration *BinaryTemporalRelationshipType*): *Enabling*, *Enabling with information passing*, *Suspend-Resume*, *Task-Independence*, *Disabling*.
- A set of tasks composes a *Task Taxonomy*. A task only can belong to one task taxonomy.
- Finally, notice that the element Task has been defined as abstract (its name is represented with italic characters). This means that we cannot create a task element in the task-based requirements model. We can only create elementary tasks or non-elementary tasks.

Furthermore, the following constraints need to be considered in order to correctly define a task taxonomy:

Constraint 1. Two tasks cannot have the same name. In OCL:

Context Task

```
Inv: self.allInstances -> forAll(t1, t2 | t1<>t2  
implies t1.Name <> t2.Name)
```

Appendix A

Constraint 2. Binary Relationships can only be used to relate tasks that are both child of a same parent task. In OCL:

```
Context Binary Relationship
  Inv: self.sourceRelation.parentTask=
      self.targetRelation.parentTask
```

Constraint 3. Unary Relationships can only be used to relate a task with itself. In OCL:

```
Context Unary Relationship
  Inv: self.sourceRelation = self.targetRelation
```

Constraint 4. When a task is refined only one type of refinement can be used to obtain child tasks. In OCL:

```
Context Task
  Inv: self.allInstances -> forAll(t1, t2 | t1<>t2
      and t1.parentTask = t2.parentTask
      implies
          t1.TaskRefinement[isChildOf].Type=
              t2.TaskRefinement[isChildOf].Type)
```

Constraint 5. For each task refined by using a temporal refinement its subtasks must be related by means of a temporal relationship. In OCL:

```
Context Task
  Inv:
  self.TaskRefinement [isChildOf].Type ='Structural'
xor
  (
  self.TaskRefinement [isChildOf].Type ='Temporal'
      and
          ( self.sourceRelation->notEmpty() or
              self.targetRelation->notEmpty() )
  )
```

A3 Task Performance Description Package

Figure A3 shows the class diagram that defines both the elements and the relationships between elements of the Task Performance Description package. This package is defined as follow:

Appendix A

- Each task of the task taxonomy (see package Task Taxonomy) is associated to a *Performance Description*. Only one performance description is associated to a task. Two tasks cannot share a same performance description.
- A Performance Description is defined from a set of *Arcs* and a set of *Nodes*. There are two special nodes: an *initialNode* and a *finalNode*.
- There are Nodes of two types: *Interaction Points* and *System Actions*. Interactions Points present the attribute *Entity* that indicates the entity to which they are associated. Interaction Points are associated to an element of the package System Information that describes the information that is shown or requested in each Interaction Point in detail.
- There are Interaction Points of two types: *Output IPs* and *Input IPs*. Output IPs present an attribute *Cardinality* that indicates the number of instances that are shown in this IP.
- There are System Actions of two types: *Function SA* and *Search SA*. Function SA actions present the attribute *Name* that indicates the name of the action. Search SA actions present the attribute *Entity* that indicates the entity to which the information to be found is related.
- There are Arcs of two types: *UserAction* and *SequenceOfNodes*. A *SequenceOfNodes* presents the attribute *Condition* that indicates the condition that must be satisfied for activating the sequence.
- *SequenceOfNodes* are classified into two types: *SequenceOfActionIP* and *SequenceOfActions*.
- A *SequenceOfActionIP* connects a System Action (arc's source) to an Output IP (arc's target). A *SequenceOfActions* connects a System Action (arc source) to another System Action (arc's target).
- There are UserActions of three types: *IntroductionOfInformation*, *ActivationOfOperation* and *SelectionOfInformation*.
- An *IntroductionOfInformation* connects an Input IP (arc's source) to a System Action (arc's target). An *ActivationOfOperation* connects an Output IP (arc's source) to a System Action (arc's target). A *SelectionOfInformation* connects an Output IP (arc's source) to another Output IP (arc's target).
- The following elements have been defined as abstract elements (its name is represented with italic characters): Arc, SequenceOfNodes, UserAction, Node,

Appendix A

InteractionPoint and SystemAction. This means that these elements cannot be created a in the task-based requirements model.

Furthermore, the following constraints need to be considered:

Constraint 1. Two OutputIPs cannot have the same entity and the same cardinality. In OCL:

```
Context Output IP
  Inv: self.allInstances -> forAll(o1, o2 | o1<>o2
    implies o1.Entity <> o2.Entity
    or
    o1.cardinality <> o2.cardinality)
```

Constraint 2. Two FunctionSA actions cannot have the same name. In OCL:

```
Context Function SA
  Inv: self.allInstances -> forAll(s1, s2 | s1<>s2
    implies s1.Name <> s2.Name)
```

Constraint 3. InputIPs cannot be neither initialNodes nor finalNodes. In OCL:

```
Context InputIP
  Inv: self.initialNode->isEmpty() and
    self.finalNode->isEmpty()
```

Constraint 4. SearchSA actions cannot be finalNode. In OCL:

```
Context SearchSA
  Inv: self.finalNode->isEmpty()
```

Constraint 5. The attribute Type of an ActivationOfOperation must be only defined when the arc's target is a FunctionSA. In OCL:

```
Context ActivationOfOperation
  Inv: self.target.oclIsTypeOf(FunctionSA) xor
    self.Type.size()=0
```

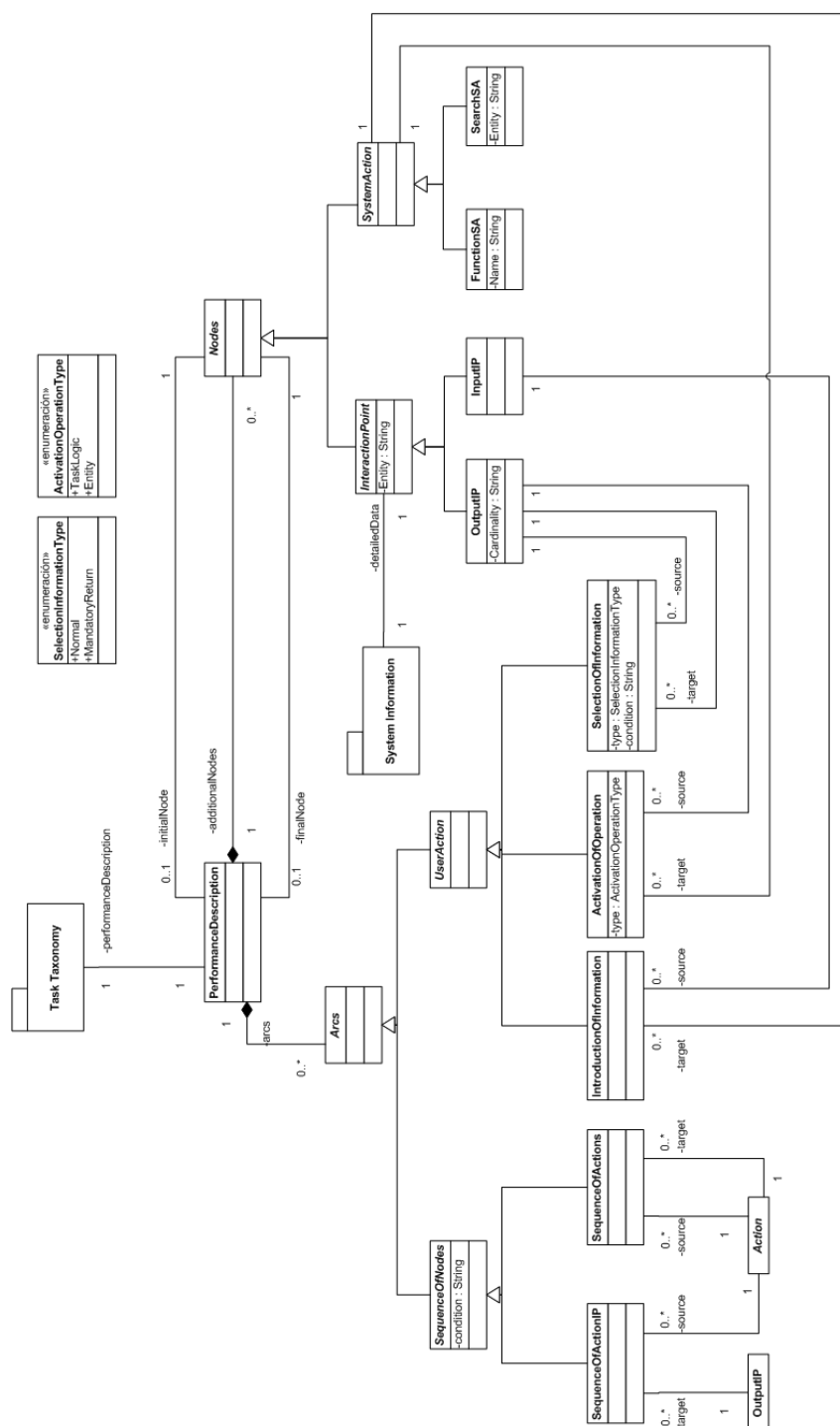


Figure A3 Class diagram of the Task Performance Description package.

4 System Information Package

Figure A4 shows the class diagram that defines both the elements and the relationships between elements of the System Data package. This package is defined as follows:

- On the one hand, we can find the element *Information Template*. Each Information Template presents two attributes: *Id* which indicates its identifier and *Entity* which indicates the entity to which the template is associated.
- An Information Template can be the *superType* of other presentation templates. A presentation template only can be *subType* of one template.
- An information template is described by a set of *DataElements*. Each DataElement can be associated to one template. They present two attributes: *Name* which indicates the name of the element and *Description* which specifies a description of the element.
- There are two types of DataElements: *SimpleNatureDE* and *ComplexNatureDE*. Each SimpleNatureDE presents an attribute *SimpleNature* which indicates the nature of the SimpleNatureDE. Each ComplexNatureDE is associated to the information template that describes it. A ComplexNatureDE only can be described by an Information Template. An Information Template can describe many ComplexNatureDE.
- On the other hand, we can also find the element *ExchangedDataTemplate*. This element presents two attributes: *Id* which indicates its identifier and *RetrievalCondition* which indicates a constraint for the information retrieval.
- Each ExchangedDataTemplate is associated to an Interaction Point defined in a task performance description (see package Task Performance Description).
- An ExchangedDataTemplate may present one or more *ExchangedDataSections*. We allow creating more than one ExchangedDataSections in order to support those IPs that are associated to entities that are super-types of other entities (see for instance the IP Output(Product,1) that is defined in the elementary task Inspect Shopping Cart (see Figure 4.21); Product is a super-type of CD, Software and Book (see Figure 4.25)). In these cases, the exchanged data section is defined from features of the sub-entities. For each sub-entity we define an ExchangedDataSection.
- Each ExchangedDataSection is defined from a set of *ExchangedElements*. Each ExchangedElement is associated to one or more SimpleNatureDE. We

Appendix A

associate an ExchangedElement to a set of SimpleNatureDE for allowing the exchanged of features that has a complex nature and are associated to a super-type (see for instance Figure 4.28). A SimpleNatureDE can be associated to many ExchangedElement.

- Finally, notice that the element DataElement has been defined as abstract (its name is represented with italic characters). This means that we can only create ComplexNatureDE or SimpleNatureDE.

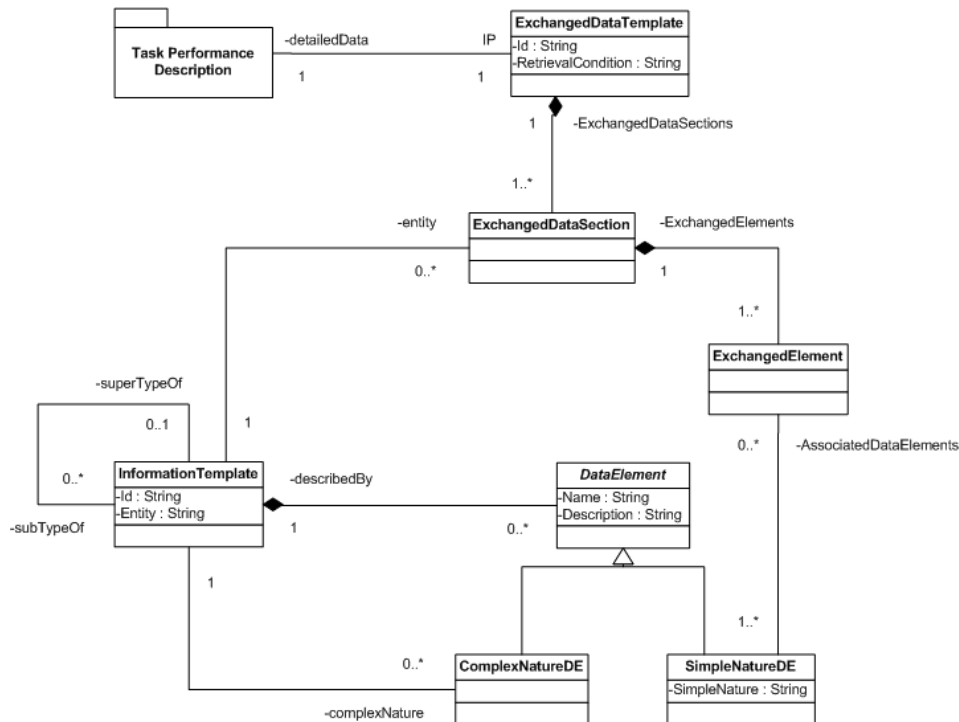


Figure A4 Class diagram of the System Information package.

Furthermore, in order to correctly define the information that the system must store the following constraints need to be considered:

Constraint 1. Two InformationTemplates cannot have the same Id. In OCL:

```
Context InformationTemplate
  Inv: self.allInstances -> forAll(t1, t2 | t1<>t2
    implies t1.Id <> t2.Id)
```

Appendix A

Constraint 2. Two ExchangedDataTemplates cannot have the same Id. In OCL:

Context ExchangedDataTemplate

```
Inv: self.allInstances -> forAll(d1, d2 | d1<>d2
    implies d1.Id <> d2.Id)
```

Constraint 3. Two DataElements associated to a same InformationTemplate cannot have the same name. In OCL:

Context DataElement

```
Inv: self.allInstances -> forAll(e1, e2 | e1<>e2
    and e1.describedBy=e2.describedBy
    implies e1.Name <> e2.Name)
```

Constraint 4. An InformationTemplate may be associated to no DataElements only if it is a supertype of other InformationTemplate. In OCL:

Context InformationTemplate

```
Inv: self.describedBy.notEmpty()=true or
    self.superTypeOf->notEmpty()=true
```

Constraint 5. ExchangedElements of an ExchangedDataSection must be associated to: (1) DataElements that belong to the entity to which the ExchangedDataSection is associated or (2) DataElements that belong to an entity that constitutes the nature of a complex nature attribute of the entity associated to the ExchangedDataSection. In OCL:

Context ExchangedElement

```
Inv:
    self.associatedDataElements->forAll(
        de |
            de.InformationTemplate =
                self.ExchangedDataSection.InformationTemplate
                or
                de.InformationTemplate.subTypeOf=
                    self.ExchangedDataSection.InformationTemplate
    )
```


Appendix A

Constraint 6. If an `ExchangedElement` is associated to more than one `SimpleNatureDE`, `SimpleNatureDEs` belongs to different sub-entities of the entity associated to the `ExchangedDataSection`. In OCL:

Context `ExchangedElement`

Inv:

```
self.SimpleNatureDE->size() = 1
xor
self.SimpleNatureDE->forall(
    de1, de2: SimpleNatureDE,
    t1, t2: InformationTemplate
    |
    de1<>de2 and
    t1=de1.InformationTemplate and
    t2=de2.InformationTemplate

    implies
    t1<>t2 and
    t1.subTypeOf=
        self.ExchangedDataSection.InformationTemplate
    t2.subTypeOf=
        self.ExchangedDataSection.InformationTemplate
)
```

Constraint 7. If an `ExchangedDataTemplate` has an only one `ExchangedDataSection`, the entity (`InformationTemplate`) associated to the `ExchangedDataSection` must be the same as the entity associated to the IP for which the `ExchangedDataTemplate` is defined.

Context `ExchangedDataTemplate`

Inv:

```
self.ExchangedDataSection->size()=1

and

self.ExchangedDataSection.InformationTemplate.
Entity= self.InteractionPoint.Entity
```

Appendix A

Constraint 8. If an `ExchangedDataTemplate` has more than one `ExchangedDataSection`, the entity associated to each `ExchangedDataSection` must be a sub-entity of the entity associated to the IP for which `ExchangedDataTemplate` is defined

Context `ExchangedDataTemplate`

Inv:

```
self.ExchangedDataSection->size() > 1
```

and

```
self.ExchangedDataSection->forall(e |  
e.InformationTemplate.subTypeOf.Entity=  
self.InteractionPoint.Entity)
```

Appendix B

The OOWS Approach in a Nutshell

B1 Introduction

OOWS (Object Oriented Web Solutions) is the extension of the object-oriented software production method OO-Method ([Pastor et al 2001]) that introduces the required expressivity to capture the navigational and presentational requirements of Web applications.

OO-Method provides model-based code generation capabilities and integrates formal specification techniques with conventional OO modelling notations. The OO-Method conceptual schema allows specifying the structural and behavioural aspects of traditional applications by means of three models:

- A *Structural Model*, which defines the system structure from a set of classes (with their operations and attributes) and a set of relationships between classes (specialization, association and aggregation). To do this, a UML Class Diagram [UML] is used.
- A *Dynamic Model*, which describes the different valid object-life sequence for each class of the system using State Transition Diagrams. Also in this model object interactions (communications between objects) are represented by Sequence diagrams.
- A *Functional Model*, which captures the semantics of state changes to define service effects by using a textual formal specification.

Appendix B

OOWS [Fons et al. 2003] extends the OO-Method conceptual schema by introducing two new models in order to deal with navigation specification and Web user interface definition. These models are:

- A *Navigational Model*, which captures the navigational structure of Web applications. This model defines the system user types and how they access the system information and functionality.
- A *Presentation Model*, which defines the presentation properties in which the information and functionality must be shown in the Web application.

Furthermore, the code generation strategy of OO-Method is also extended by introducing new transformation patterns that supports the abstract primitives incorporated by these two new models.

In this thesis, we use the OOWS/OO-Method conceptual schema and its associated code generation strategy in order to obtain Web application prototypes from requirements specifications. In particular, we use both the structural model and the navigational model, which provide enough information to construct Web application prototypes. The structural model is based on the well-known primitives proposed in the UML 2.0 class diagram [UML]. The navigational model is explained in Section B2 of this appendix. Section B3 introduces moreover the meta-model of the OOWS navigational model in order to provide a more precise definition of it. Section B4 presents an overview of the extensions to the code generation strategy that are incorporated by the OOWS method. Finally, Section B5 presents the OOWS CASE tool that supports this method.

B2 The OOWS Navigational Model: an Overview

The OOWS navigational model is made up of two diagrams: the *user diagram* and the *navigational diagram*. In order to introduce both diagrams we use the Amazon example presented throughout this thesis.

B2.1 User Diagram

The user diagram allows us to express the type of users (user roles) that can interact with the Web application. This diagram provides mechanisms to properly cope with additional user management capabilities, such as user specialization which allows defining user taxonomies to improve navigational specification reuse (see Figure B1) [Fons et al. 2002].

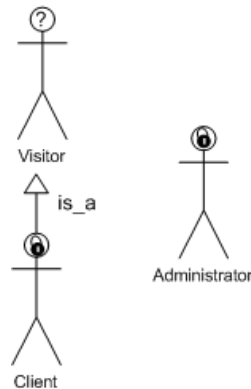


Figure B1 User Diagram of the Amazon Example

There exist different types of user roles depending on how they connect to the system. In Figure B1 we can distinguish the following two types of user roles:

- *Anonymous user roles* (depicted with a ‘?’ in his head): these user roles do not need to provide information about their identity. In the Amazon example, we have defined an anonymous user role: *Visitor*.
- *Registered user roles* (depicted with a lock in his head): these user roles need to be identified to connect to the system. In the Amazon example, two registered user roles are defined: *Client* and *Administrator*.

B2.2 Navigational Diagram

Once user roles have been identified, a structured and organized system view for each user role must be specified. These views are defined over the class diagram, in terms of the visibility of class attributes, operations and relationships.

We capture the navigation specification in two steps: the *Authoring-in-the-large* (global view) and the *Authoring-in-the-small* (detailed view).

The *Authoring-in-the-large* step refers to the specification and design of global and structural aspects of the Web application. It is achieved by defining both a set of system-user abstract interaction units and how the user can navigate from one to another. These requirements are specified in a **Navigational Map** that provides the system view and accessibility for each user role. It is represented using a directed graph whose nodes are navigational contexts (forward defined) and its arcs denote navigational links or valid navigational paths (see Figure B2).

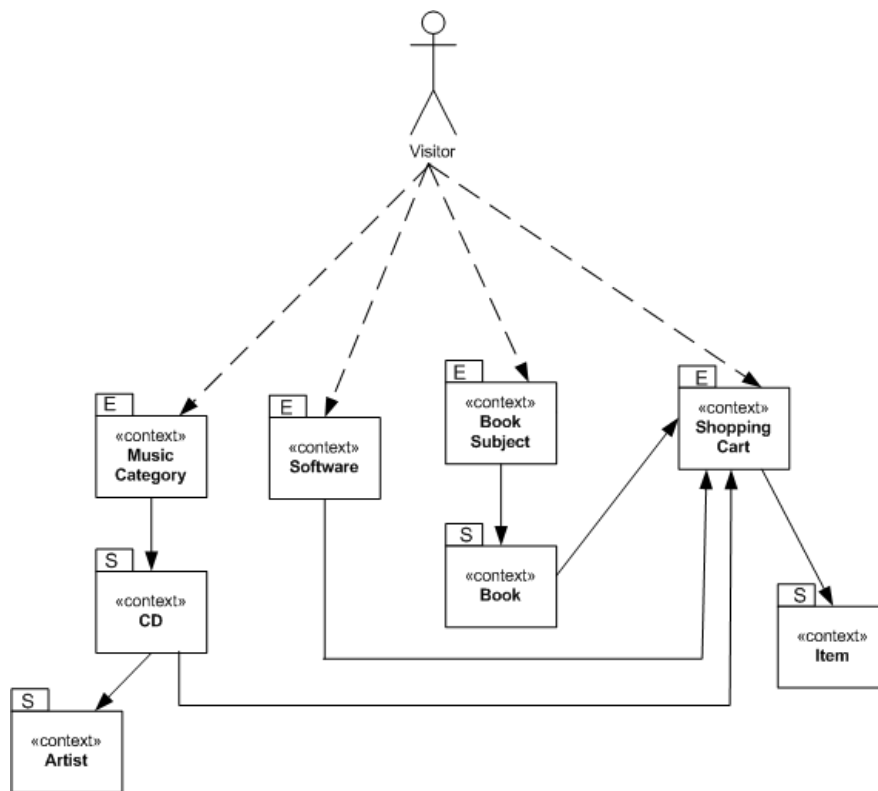


Figure B2 Navigational Map of the Amazon Example

These **navigational contexts** (graphically represented as UML packages stereotyped with the «context» keyword) represent user interaction units that provide a set of cohesive data and operations to perform certain activity. In order to define the context reachability, we propose contexts of two types:

- Exploration navigational contexts (depicted with the “E” label): which are reachable nodes from any node²¹. These contexts define implicitly navigational links from any node and explicitly (using dashed arrows) from the root of the map represented by the user role (see Figure B2, contexts Music Category, Software, Book Subject, etc.).
- Sequence navigational contexts (depicted with the “S” label): which can only be accessed via a predefined navigational path by selecting a sequence link (forward defined). For instance, the context Artist (see Figure B2) can only be reached following the path Music Category-CD-Artist.

²¹ Similar to the *Landmark* pattern in the Hipermedia Community

Appendix B

Navigational links (navigational map arcs) represent context reachability or “*navigational paths*”. There are two types of navigational links:

- *Sequence links* or “*contextual links*” (represented with solid arrows) define a *semantic* navigation between contexts. Selecting a sequence link implies carrying contextual information to the target context (e.g. the object that has been selected in the source navigational context).
- *Exploration links* or “*non contextual links*” (represented with dashed arrows) represent a user intentional change of task. When an exploration link is crossed, no contextual information is carried to the target context.

In order to cope with complex navigational models, navigational maps are structured by introducing *navigational subsystems*. A navigational subsystem is a primitive that allows defining a sub-graph within the full graph (hyper graph). Recursively, the content of a subsystem is defined by means of another navigational map.

The *Authoring-in-the-small* step refers to the detailed specification of the contents of the navigational contexts. To specify this content, each navigational context is made up of a set of **navigational classes** that represent class views (including attributes and operations). These classes are stereotyped with the «view» keyword (see Figure B3). Each navigational context has one mandatory navigational class, called manager class and optional navigational classes to provide complementary information of the manager class, called complementary classes.

Figure B3 shows the navigational context CD. The purpose of this context is to provide detailed information about a CD. To do this, the following navigational classes have been defined in this context: the class CD which defines the manager class and the classes Artist and Music Category which define complementary classes. These classes include the subset of attributes and operations that users can see or activate in this context.

All navigational classes must be related by unidirectional binary relationships, called **navigational relationships**. They are defined over existing aggregation/association/composition or specialization/generalization relationships representing the retrieval of the related instances by these relationships. Two kind of navigational relationships can be defined, depending on whether they define a navigation capability or not:

- A context dependency relationship (graphically represented by dashed arrows) represents a basic information recovery by crossing a structural relationship between classes. When a context dependency relationship is defined, all the instances of the target class which are related to the origin class are retrieved. Relationships of this kind do not define any navigation. Figure B3 shows a

Appendix B

context dependency relationship between the classes CD and Music Category. This relationship is specifying that the context retrieves for each CD the name of the music category to which the CD belongs.

- A context relationship (graphically represented by solid arrows) represents the same information recovery as a context dependency relationship does, plus a navigation capability to a target navigational context, creating a sequence link in the navigational map. They have the following properties:
 - A context attribute that indicates the target context of the navigation (depicted as [targetContext]).
 - A link attribute that specifies the attribute used as the “anchor” to activate the navigation to the target context. This attribute can be an attribute of either the source navigational class or the target navigational class.

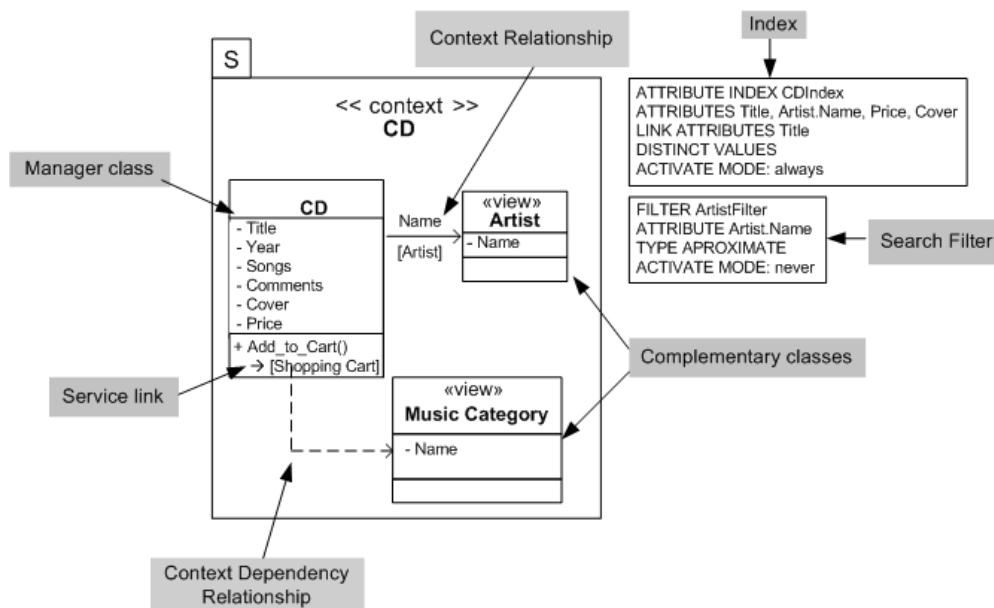


Figure B3 Navigational Map of the Amazon Example

Figure B3 shows a context relationship between the classes CD and Artist. This relationship is specifying that the context retrieves for each CD the name of the artist that has recorded it. Furthermore, this contextual relationship creates a link to the target context (Artist), by using the link attribute (name of the Artist) as the “anchor”, and carrying information about the selected artist. Thus, users can access the context

Appendix B

Artists by selecting the name of the artist in order to obtain more detailed information about the CD's artist.

Service links can also be attached to a class operation. A service link represents a navigational context that users must reach after the operation execution. Figure B3 shows a service link attached to the operation *Add_to_Cart()* of the ManagerClass. This service link specifies that after the execution of *Add_to_Cart()*, the system must automatically navigate to the navigational context Shopping Cart. There are two types of service links:

- *Contextual Service Links*: They navigate to the target context by carrying contextual information to the target context. This contextual information is the object to which the executed operation belongs. They are represented as follow:

method() →[target context]

For instance, when users activate the operation *Add_to_Cart()*, they access the navigational context Shopping Cart. Furthermore, the CD which is added to the Shopping Cart (this one to which the executed operation belongs) also navigates to this context.

- *Non-Contextual Service Links*: They navigate to the target context without carrying contextual information. They are represented as follow:

method() [target context]

Finally, we can also define **information access mechanisms** for each navigational context. Navigational contexts retrieve the population of classes of the conceptual model. We define the cardinality of a navigational context as the number of instances that it retrieves. Sometimes the retrieved information is difficult to manage. To help users browsing this amount of information, it is necessary to define mechanisms for browsing and filtering this information. There exist two main information access mechanisms: Search Filters and Indexes.

- *Search filters*, which allow us to filter the space of objects that retrieve the navigational context. There are three types of filters: (1) *exact* filters which take one attribute value and return all the instances that match it exactly; (2) *approximate* filters which take one attribute value and return all the instances whose attribute values include this value as a substring; and (3) *range* filters take two values (a maximum and a minimum) and return all the instances whose attribute values fit within the range.

For instance, the CD navigational context presents a search filter which allows users to find all the CDs of a specific artist (see Figure B3).

Appendix B

- *Indexes*, which are structures that provide an indexed access to the population of objects. Indexes create a list of summarised information that allows users to choose one item (instance) from the list. This selection causes this instance to become active in the navigational context.

For instance, the navigational context in Figure B3 provides users with a list of summarised information where the title, the artist's name, the price, and the cover are shown for each CD.

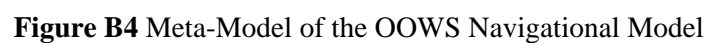
There are three ways in which the access mechanisms defined in a navigational context can be activated:

- *Always*: the access mechanisms are automatically activated when users access the navigational context. It means that when users access the navigational context they must interact with the access mechanisms instead of obtaining the information defined in the navigational context. This information will be retrieved only for the instances that users select from the access mechanism.
- *Never*: the access mechanisms must be manually activated by users. It means that when users access the navigational context they obtain the information defined in it. Next, the user can activate the access mechanisms in order to filter the provided information.
- *By Threshold*: this type of activation is a mixture between the two previous. If the navigational context retrieves more than a specific number of instances then the access mechanisms are automatically activated (always). Otherwise, users must manually activate the access mechanisms (never).

See [Fons *et al.* 2003] for more detailed information about the OOWS method.

B3 Meta-Model of the OOWS Navigational Model

In order to provide a more precise definition of the OOWS navigational model its meta-model is shown in Figure B4. This meta-model is defined according to the Meta Object Facilities (MOF) [MOF]. In order to specify the elements of the meta-model and the relationships between them we use a class diagram.



B4 The OOWS Code Generation Strategy

In order to develop the Web application, we define a two step process that (1) proposes a multi-tier architectural style, based on a classical three tier architecture, and (2) defines a set of correspondences between the conceptual abstractions and the software elements that implement each tier of the architecture, making use of design patterns. The tiers of the selected architectural style are the following:

- Presentation Tier. It includes the graphical user interface components for interacting with the user.
- Application Tier. This tier implements the structure and the functionality of the classes in the conceptual schema.
- Persistence Tier. It implements the persistence and the access to persistent data in order to hide the details of data repositories to upper tiers.

The Application and Persistent tiers are generated from the structural model and the functional and dynamic models. Information about the correspondences between the abstractions of these models and the software elements of the tiers can be obtained in [Pastor *et al.* 2001].

The interface tier is generated from the navigational and presentation model. The navigational model allows us to generate a set of interconnected Web pages that constitutes the navigational structure of a Web application. The presentation model indicates the presentation properties of the data that is provided in each Web page. In this thesis, we focus on the generation of code from the navigational model only. Since presentation aspects are not considered in the navigational model, Web pages are generated in plain text. However, presentation aspects can be further applied by means of CSS templates. In fact, we have applied a default presentation template to the Web pages presented further in order to better visualize them. Information about the presentation model can be found in [Fons *et al.* 2003].

B4.1 Deriving the Web Application Navigational Structure from the Navigational Model

Starting from the navigational model, a group of connected Web pages for each kind of user can be obtained in a systematic way. These Web pages define the Web application user interface for navigating, visualizing the data and accessing to the Web application functionality. As a representative example, part of the Web pages that are obtained for the Visitor user in the Amazon example (see Figure B2) are next introduced.

Appendix B

A Web page is created for each navigational context in the navigational map. Each of these Web pages is responsible for retrieving the specified information in its navigational context by requesting it to the application tier. As home page, a Web page that provides a link to each navigational exploration context (page) is created. Figure B5 shows an example of a created home page. Clicking on a link of the menu (situated at the left side of the figure), the application navigates to the Web page that represents the target context. For instance, if we select the Music Category link in the home page, we reach (using an exploration link) the Music Category Web page (related to the Music Category context). In this page, we obtain information about music categories.



Figure B5 Created Home Web Page

If we select one category the page CD is accessed. This page implements the navigational context CD. As this context has defined an index (whose activation mode is “always”), when we reach the Web page, the index gets activated, creating a list of instances with the specified information. This page is shown in Figure B6. This list of instances is defined from the list of CDs that belongs to the selected music category (the music category is passed to the context CD since it is accessed throughout a sequence exploration link).

Selecting one of these indexed instances (using the title of the CD as the link attribute), the Web page in Figure B6 is accessed. This page shows all the information specified in the context CD. The web page of Figure B7 presents the CD context. It shows information about the CD (title, year, song, comments, cover and price) as well as about its artist (name) and its music category (name).

Appendix B

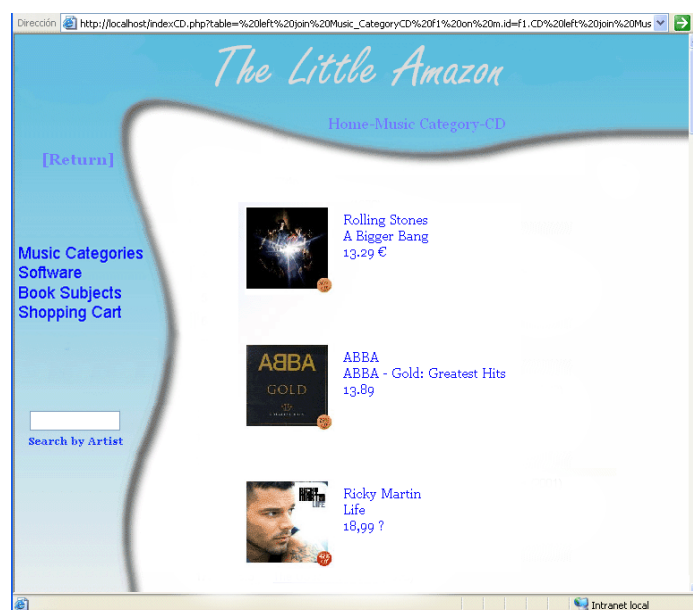


Figure B6 Web Page for the index of the navigational context CD



Figure B7 Web Page for the navigational context CD

B5 Tool Support

The OOWS method is supported by the OOWS CASE tool that provides support for two main aspects: Model Management and Generation of code. The last version of this tool can be found in [Valverde *et al.* 2007].

B5.1 Model Management

By *support for Model Management* we mean creating models in a visual way as well as for storing models in a persistent way

The OOWS CASE tool provides us with a graphical editor that allows us to create OOWS conceptual models. Figure B8 shows a snapshot of this graphical editor. In the upper side of this figure, we can see the modeling frame where the OOWS models can be defined by using the mechanism provided by the toolbar situated at the right side. In the lower side of the figure, we can see the property frame. In this frame we can specify the properties of the different primitives created in the modeling frame.

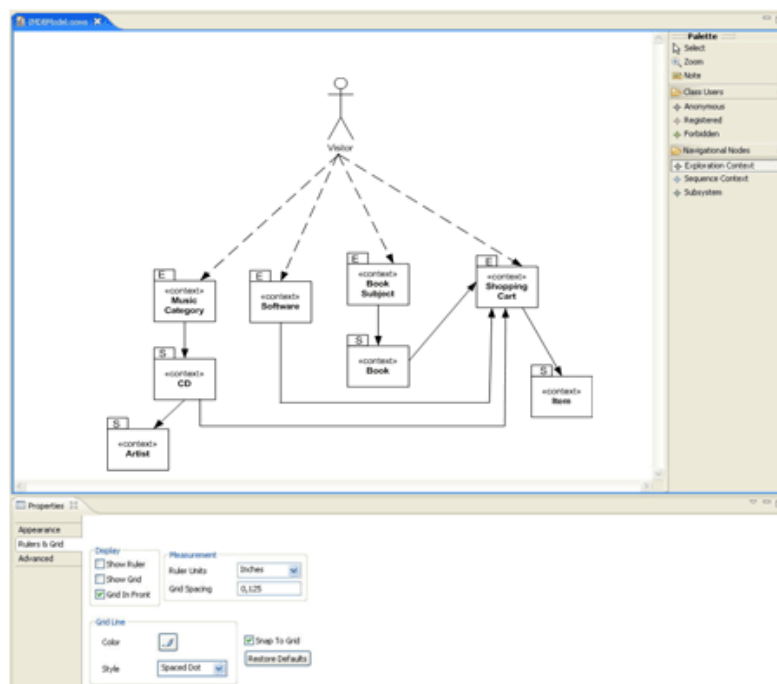


Figure B8 OOWS Modelling tool

Appendix B

The OOWS CASE tool provides us with a mechanism to make models persistent based on XML. A proprietary XML language has been defined in order to store/load OOWS conceptual models into/from XML documents. Figure B9 shows a representative example of this document.



Figure B9 Example of XML code for storing OOWS models

An OOWS XML document is divided into two main sections: *ClassDiagram* that stores the structural model and *NavigationalModel* that stores the navigational model.

The class diagrams section is made up of labels such as *Class*, *Attribute*, *Operation*, *Transaction* or *Relationship*. These labels are used to store the different elements of the structural model.

The navigational model section is divided in two subsections:

Appendix B

- (1) *NavMap* where the navigational maps are described by indicating which navigational contexts (label *NavContext*) and navigational subsystems (label *NavSubsystem*) are defined.
- (2) *NavNodesD* where the different navigational contexts are defined. The *NavNodesD* section is made up of labels such as *NavContext_D*, *ManagerNavClass_D*, *ContextDepRel_D*, *ComplementaryNavClass_D*, *NavAttribute*, *NavOperation*, *index*, *filter*, and so on.

B5.2 Generation of Code

As far the *Generation of Code*, the OOWS CASE tool provides us with two modes:

- *Full application mode*: By means of this mode, a generation engine obtains code that implements fully operative Web applications. To achieve this, the generated code is prepared to be integrated with the code generated by the commercial tool Olivanova [Olivanova]. On the one hand, the Olivanova tool generates the Application and the Persistent tiers of the Web application from the structural model, and the functional and dynamic models. On the other hand, the OOWS case tool generates the Interface tier of the Web application from the navigational and the presentation model. The code of the Interface tier is implemented by means of a high level implementation framework. This framework is in charge of connecting the Interface tier generated by the OOWS case tool with the Application tier generated by the Olivanova tool. See [Valverde *et al.* 2007] for additional information about this code generation strategy.
- *Prototyping mode*: This mode of code generation is provided in order to use the OOWS case tool without the commercial tool Olivanova. In this mode, another generation engine is used. It automatically implements a Web application prototype from the structural model and the navigational model. In this case, the generation engine implements the Web application interface as well as the data base schema in order to store the Web application data. Furthermore, code to connect the Web application interface with the database is also generated. This code implements the retrieval of data defined in each navigational context as well as CRUD²² operations for this data. In this generation mode, the PHP technology is used to implement the prototype. The OOWS tool generates a set of PHP files which produces HTML code. This code implements Web pages as plain text. Presentation aspects can be incorporated by means of CSS templates. This is the generation mode used for the purposes of this thesis.

²² Create/Read/Update/Delete

Appendix B

In Figure B10 we can see a snapshot of the generation engine that allows us to obtain Web application prototypes. This generation engine presents a very intuitive and easy to use interface. In order to generate a Web application prototype we just need to do the following steps (see Figure B10):

1. To select the XML document where the OOWS conceptual model is stored.
2. Optionally, to select a predefined presentation template that associates aesthetic aspects (such as colours, font face or element positions) to the generated code.
3. And finally, to indicate meta-information such as the title of the Web application and a welcome message.

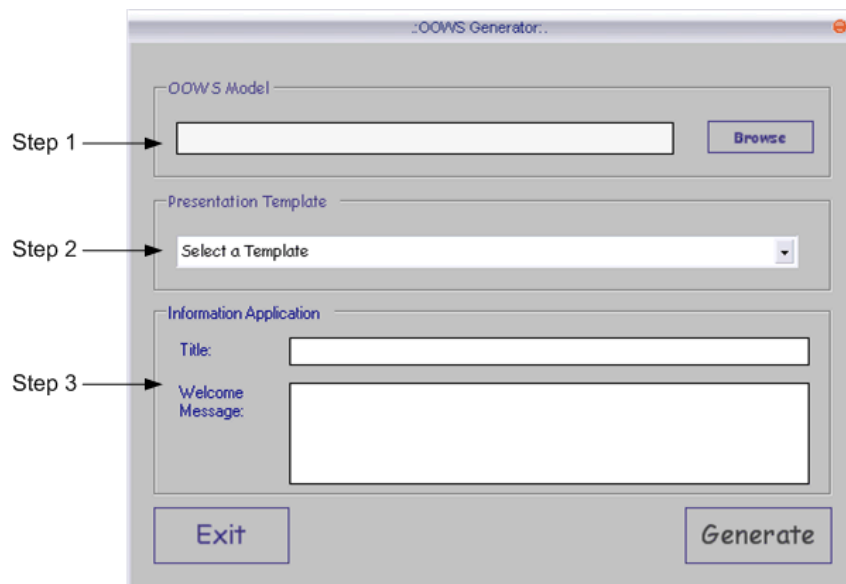


Figure B10 OOWS Generation Engine for the Prototyping mode

Figure B11 shows the structure (folders and files) of a Web application prototype generated by the prototyping generation engine. This structure is divided in two main folders: Application and DB. The last one contains the SQL script that allow us to create the data base required for the Web application. The folder Application contains the files that implement the Web application prototype and which connect to the data base created by the SQL script.

In the folder Application, we can find several PHP files as well as several subfolders. The PHP files implement the navigational structure of the Web application prototype. These files are generated from the navigational contexts defined in the OOWS

Appendix B

navigational model as well as from the access mechanisms (indexes and search filters). These files are in charge of providing users with the proper information (according to the definition of navigational contexts) as well as of providing mechanisms that allow users to navigate the whole Web application navigational structure.

In addition of these PHP files, we can find the following subfolders inside the Application folder:

- (1) The *css* subfolder, which contains the presentation template (defined as a set of css styles) that have been selected in the generation engine (step 2, see Figure B10).
- (2) The *Image* subfolder, which is initially an empty folder but which is created to place the images that the Web application prototype needs.
- (3) The *Classes* subfolder, which contains several PHP files that implement PHP classes with the structure of the classes defined in the OOWS structural model. These PHP classes implements the class attributes as well as the signature of the class operations. They are created in order to constitute a base for manually implementing the functionality that is not automatically generated.
- (4) The *InfoQueries* subfolders, which contains both the PHP file *infoqueries.php* that presents functions that retrieve the information required for each PHP file (e.g. for the PHP file that provides the description of a CD, exists a function that queries the database obtaining the required information) and *connection_bd.php* that facilitates the connection to the database as well as the execution of queries.
- (5) The *Support* subfolder, which contains PHP files that allow us to configure the Web application prototype in run time. These files are the following: *alias.php* that configures the text that appear in menus, heads, foots etc; *bd.php* that indicates the name of the database and facilitates us to change it; and *template.php* that indicate the presentation template (which must be located in the folder *css*) that is being used (this file facilitates us to change the look and feel design of the Web application in run time).

Appendix B



Figure B11 Structure of the generated prototype

According to the files shown in Figure B11, the architecture of the Web application prototype is the following (see Figure B12): The Web interface that is accessed from a Web browser is implemented by means of HTML code. This code is produced by the PHP files included in the Application folder (those that constitute the navigational structure). The produced HTML code structures the information that must be provided to users without considering aspects related to images or aesthetic properties. These aspects are considered by introducing into the HTML code references to the files included in the folders *Images* and *css*. The PHP files obtain the information from the database by using the functions provided by the file *infoqueries.php*. This file presents functions that retrieve the information required for

Appendix B

each PHP files from the database. To do this, the auxiliary file *connection_bd.php* is used for connecting to the database. After obtaining the corresponding information from the data base, the file *infoqueries.php* send the information to the PHP files which format it into HTML code. In order to facilitate the configuration of this HTML code in run time files included in the folder *Support* are used.

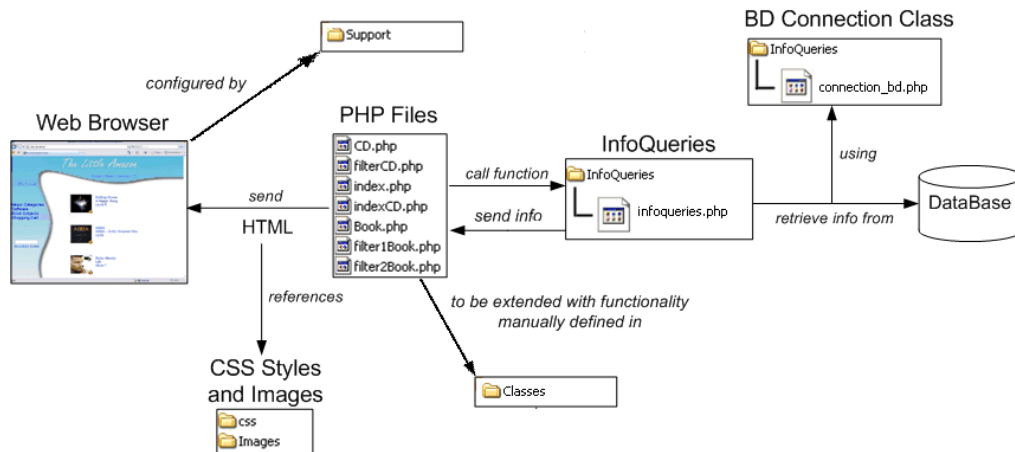


Figure B12 Architecture of the generated prototype

Figures B5, Figure B6 and Figure B7 show some examples of the Web pages produced by the PHP files generated by the OOWS prototype generation engine.

Appendix C

Some Essentials of Graph Transformations

C1 Introduction

Graphs are a powerful and well-established tool to represent several data structures as well as states of concurrent and distributed systems. In a general way, a graph can be used to describe whole sets of objects including the relations between them [Kreowski et al. 2006]. In this thesis, we use graphs in order to represent task-based requirements models and OOWS conceptual models. In particular, we use identified, labelled, typed, constrained and direct graphs.

Graphs are used not only to represent static structures but also to represent dynamical ones. To do this, a graph which is able to transform is needed. In this context, several algebraic approaches have been introduced to transform graphs. One of these approaches is the Single Push Out approach (SPO) which is used in the context of this thesis.

Section C2 introduces some notions about graph theory that are required for understanding the topic of graph transformations. Section C3 introduces algebraic graph theory as technique to perform graph transformations.

C2 Notions about Graph Theory

In this section, some important notions about graph theory are explained. It is important to understand these notions in order to better understand how graph transformations are applied. All notions are introduced in a very intuitive way in order to facilitate its understanding. In order to access the formal foundations in which these notions are based see the works that are referenced in each sub-section.

C2.1 Basic Definitions about Graphs

Next, basic definitions about the notion of graph are introduced. In order to access formal foundations of these definitions see [Heckel 2004] [Ehrig 1979] or [Löwe 1993]. Furthermore, an application of these formal foundations in the ambit of model-to-model transformations can be found in [Limbourg 2004].

Definition 1: Graph. A graph is defined from a set V of *vertices* and a set E of *edges* that connect pairs of vertices. Furthermore, there exist two functions: a *source* function which indicates the source node of an edge and a *target* function which indicates the target node of an edge.

Definition 2: Subgraph and Supergraph. A subgraph of a graph G is a graph whose vertex and edge sets (V and E) are subsets of those of G . In the other direction, a supergraph of a graph G is a graph that contains G as a subgraph.

Figure C1 shows, in the left side, the graph G which is defined from seven vertices and five edges. The right side of the same figure shows a sub-graph of G . This sub-graph is defined from four of the seven vertices of G and four of the five edges of G .

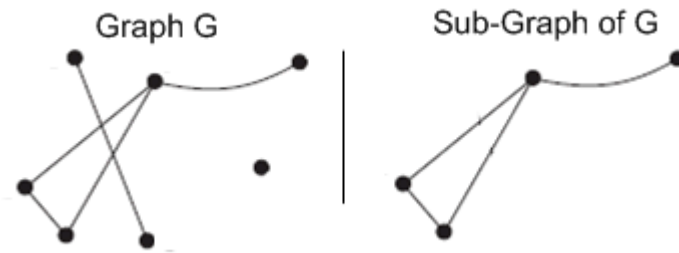


Figure C1 Example of Graph and Sub-Graph

Definition 3: Identified Graph. An identified graph is a graph for which exist functions that univocally identifies each vertex or edge of the graph. Vertices and edges are considered as individuals. In the opposite case, we find unidentified graphs which are graphs whose vertices or edges are not considered as individuals. Only the

Appendix C

way in which they connect to the rest of the graph characterize unidentified vertices and edges.

An identified graph is defined from the sets V and E , from the functions source and target, and moreover from:

- Two sets I_v and I_e of unique identifiers for vertices and edges respectively.
- Two bijective functions id_v and id_e which indicate the identifier associated to a vertex and an edge respectively.

Only identified graphs allow us to define two edges between the same two vertices, since edges are identified by themselves. In unidentified graphs, we cannot define two arcs between the same two vertices since edges are identified from the vertices that they connect. Figure C2 shows an example of Identified Graph. Vertices are identified by a natural numbers; edges are identified by characters.

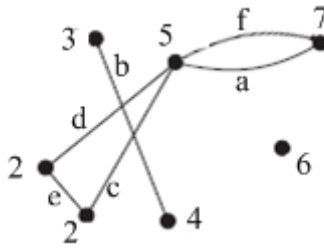


Figure C2 Example of Identified Graph

Definition 4: Direct Graph. A direct graph is a graph in which each edge symbolizes an ordered, non-transitive relationship between two vertices. Such edges are depicted with an arrowhead at one end of each edge. Figure C3 shows an example of directed graph.

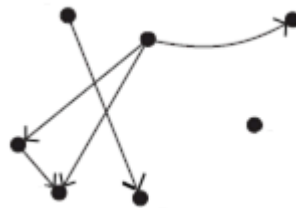


Figure C3 Example of Directed Graph

Definition 5: Labelled Graph. A labeled graph is a graph with labels assigned to its vertices and edges. These assignments do not have to be unique, i.e. different vertices

Appendix C

or edges can have the same label (not confuse labelled graphs with graph labelling [Gallian 1998]).

A labelled graph is defined from the sets V and E , from the functions source and target, and moreover from:

- Two sets L_v and L_e of labels for vertices and edges respectively.
- Two functions $label_v$ and $label_e$ which indicate the label associated to a vertex and an edge respectively.

Figure C4 shows an example of labelled identified graph. In order to not overload the graph we have only labelled two vertices and one edge. Labels are depicted with a colon after the identifier. We have associated the label “Person” to the vertex 5, the label “has” to the edge a, and the label “Car” to the edge 7. Notice that labels are many times used to refer vertices and edges (just to facilitate the reading of examples). However, they do identify neither vertices nor edges. In order to illustrate this aspect in the example, we have shown labels together with identifiers.

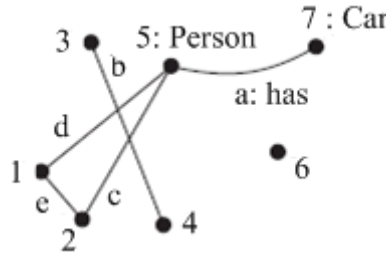


Figure C4 Example of Labelled Identified Graph

Definition 6: Typed Graph. Typing allows classifying nodes and edges by attaching types to them. A typed graph is a graph with a function that indicate the type of each graph element.. A same type may be assigned to different elements.

A typed graph is defined from the sets V and E , from the functions source and target, and moreover from:

- Two sets T_v and T_e of types for vertices and edges respectively.
- Two functions $type_v$ and $type_e$ which indicate the type associated to a vertex and an edge respectively.

Appendix C

Figure C5 shows an example of Typed Graph. In this graph there are two types of vertices and two types of edges. In order to graphically distinguish them we have depicted vertices with squares and circles and edges with dashed and solid lines.

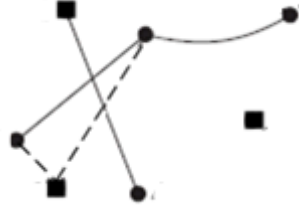


Figure C5 Example of Typed Graph

Definition 7: Constrained Graph. These graphs present constraining functions that are attached to vertices and edges. These functions associate an arbitrary number of constraints to any vertex or edge. Constraints can consist in the expression of cardinality constraints, restrictions on the domain or the co-domain of certain functions, etc. It is proposed to express these constraints with first order logic expressions.

A constrained graph is defined from the sets V and E , from the functions source and target, and moreover from:

- Two sets C_v and C_e of constraints for vertices and edges respectively.
- Two functions $constraint_v$ and $constraint_e$ which indicate the constraints associated to a vertex and an edge respectively

$$\forall v \in V, e1 \in E \mid label_v(v) = \text{"Person"} \wedge label_e(e1) = \text{"has"} \wedge source(e1) = v \rightarrow \neg \exists e2 \in E \mid label(e2) = \text{"has"} \wedge source(e2) = v$$

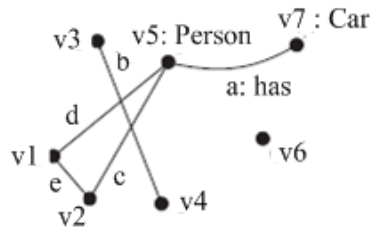


Figure C6 Example of Constrained Labelled Identified Graph

Figure C6 shows an example of Constrained Labelled Identified Graph. We present a constraint graph with labels and identifiers in order to illustrate how constraints can

Appendix C

be defined over labels (because these constraints are used in further sections). The constraint that is defined indicates that a vertex labelled with string “Person” cannot be source of more than one edge labelled with the string “has”.

Definition 8: Graphs with multiple characteristics. Graphs can present more than one of the above presented characteristic (we have already seen some example above). For instance, a graph can be labelled, constrained and direct graph; typed, identified and constrained graph; identified, typed and labelled graph; and so on. These graphs present all the characteristics that are indicated and then its definitions must be done according to these characteristics.

Let’s consider the most general case, a graph which presents all the above introduced characteristics, that is, an identified, labelled, typed, constrained and direct graph. It is defined from (graphs with other characteristic combinations are analogously defined):

- A set V of *vertices*
- A set E of *edges* that connect pairs of vertices and symbolize ordered, non-transitive relationships between two nodes.
- A function *source* which indicates the source node of an edge and a function *target* which indicates the target node of an edge.
- Two sets L_v and L_e of labels for vertices and edges respectively; and two functions $label_v$ and $label_e$ which indicate the label associated to a vertex and an edge respectively.
- Two sets I_v and I_e of unique identifiers for vertices and edges respectively; and two bijective functions id_v and id_e which indicate the identifier associated to a vertex and an edge respectively.
- Two sets T_v and T_e of types for vertices and edges respectively; and two functions $type_v$ and $type_e$ which indicate the type associated to a vertex and an edge respectively.
- Two sets C_v and C_e of constraints for vertices and edges respectively; and two functions $constraint_v$ and $constraint_e$ which indicate the constraints associated to a vertex and an edge respectively.

Figure C7 shows an example of this kind of graphs. In this thesis, we use them in order to represent models.

Appendix C

$$\forall v \in V, e1 \in E \mid \text{label}_v(v) = \text{"Person"} \wedge \text{label}_e(e1) = \text{"has"} \wedge \text{source}(e1) = v \rightarrow \\ \neg \exists e2 \in E \mid \text{label}(e2) = \text{"has"} \wedge \text{source}(e2) = v$$

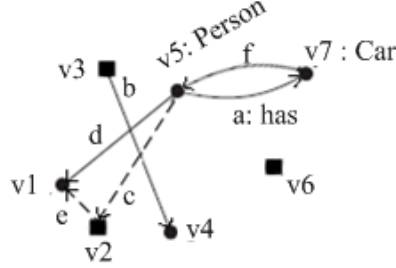


Figure C7 Example of Identified, Labelled, Typed, Direct Graph

C2.2 Graph Morphisms

The fact that a graph G occurs in another graph H is expressed by a graph morphism.

Given two graphs G and H , a morphism of G to H is defined from two mappings m_v and m_e where:

- $m_v: V_G \rightarrow V_H$ assigns vertices of G to vertices of H .
- $m_e: E_G \rightarrow E_H$ assigns edges of G to edges of H .

These mappings must satisfy the following condition:

$$\forall e_g \in E_G, e_h \in E_H \wedge m_e(e_g) = e_h \rightarrow m_v(\text{source}(e_g)) = \text{source}(e_h) \wedge \\ m_v(\text{target}(e_g)) = \text{target}(e_h)$$

This condition indicates that an edge e_g of G is assigned (by means of the mapping m_e) to an edge e_h of H if and only if the vertices which connects e_g are assigned (by means of the mapping m_v) to the vertices that connect e_h .

If m_v and m_e assign every vertex and every edge of G to a vertex or edge of H the morphism is a *total graph morphism*. Otherwise it is a *partial graph morphism*. Figure C8 illustrates an example of a total graph morphism of the graph G to the graph H .

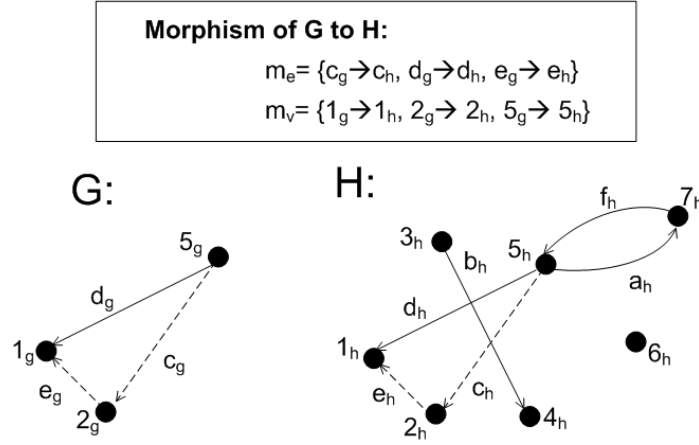


Figure C8 Example of total graph morphism.

The graph morphism presented above only preserves the graph structure, that is, nodes and edges. However, it is possible to define morphisms that preserve also other aspects such as labels, identifiers, types or constraints:

- A morphism that preserve identifiers is called an *identifier preserving (I)-graph morphism* and its definitions must be extended with the following two constraints:

$$\forall e_g \in E_G, e_h \in E_H \wedge m_e(e_g)=e_h \rightarrow id_e(e_g)=id_e(e_h)$$

$$\forall v_g \in V_G, v_h \in V_H \wedge m_v(v_g)=v_h \rightarrow id_v(v_g)=id_v(v_h)$$

- A morphism that preserve labels is called a *label preserving (L)-graph morphism* and its definitions must be extended with the following two constraints:

$$\forall e_g \in E_G, e_h \in E_H \wedge m_e(e_g)=e_h \rightarrow label_e(e_g)=label_e(e_h)$$

$$\forall v_g \in V_G, v_h \in V_H \wedge m_v(v_g)=v_h \rightarrow label_v(v_g)=label_v(v_h)$$

- A morphism that preserve types is called a *type preserving (T)-graph morphism* and its definitions must be extended with the following two constraints:

$$\forall e_g \in E_G, e_h \in E_H \wedge m_e(e_g)=e_h \rightarrow type_e(e_g)=type_e(e_h)$$

$$\forall v_g \in V_G, v_h \in V_H \wedge m_v(v_g)=v_h \rightarrow type_v(v_g)=type_v(v_h)$$

- A morphism that preserve constraints is called a *constraint preserving (C)-graph morphism* and its definitions must be extended with the following two constraints:

$$\forall e_g \in E_G, e_h \in E_H \wedge m_e(e_g)=e_h \rightarrow \text{constraint}_e(e_g)=\text{constraint}_e(e_h)$$

$$\forall v_g \in V_G, v_h \in V_H \wedge m_v(v_g)=v_h \rightarrow \text{constraint}_v(v_g)=\text{constraint}_v(v_h)$$

C2.3 Advanced Graphs

In this section, we introduce types of graph that have a special interest in the topic of graph transformations: type graphs, attributed graphs and the combination of both.

C2.3.1 Type Graphs

Type graphs were introduced as a mechanism to improve the typing of graphs [Montanari 1970] [Corradini et al. 1997] [Heckel & Wagner 1995]. The basic idea of type graphs is to define a graph whose vertices represent types for the vertices of other graphs and whose edges represent types for edges of other graphs. To do this, we must replace:

- (1) The sets of types nodes and edges (T_v and T_e , see the definition of a typed graph presented above), by a graph TG. This graph is called *type graph* and it is a labelled, constrained graph. When we use these graphs in order to represent type of nodes and types of edges, labels are used to indicate that types. Thus, the labelling functions label_v and label_e must be bijective. This aspect assures that each graph component is univocally associated to a label and that each label is associated to a graph component. This aspect is captured by means of the following two constraints:

$$\forall x, y \in (V \cup E), \text{label}(x)=\text{label}(y) \rightarrow x=y$$

$$\forall x \in (L_v \cup L_e), \exists y \in (V \cup E) \mid \text{label}(y)=x;$$

- (2) The functions type_v and type_e which indicate the type associated to vertices and edges, by a total (L,C) graph morphism.

Thus, if there is a type graph TG and also there is a total (L,C) graph morphism of a given graph G to TG, G is said to be a *TG-Typed graph*. For each vertex and edge of G there is a correspondence with a vertex or edge of TG. This correspondence preserves labels. Furthermore, constraints defined on labels in TG are also applicable on labels in G (the graph morphism is constraint preserving). Figure 9 illustrates this aspect. In order to facilitate the reading of this figure identifiers in the type graph are not shown. Furthermore, the cardinality constraint associated to the edge with the

Appendix C

label “isMarried” has been defined using the UML notation [UML]. Its definition in first order logic is analogous to the one presented in Figure C6.

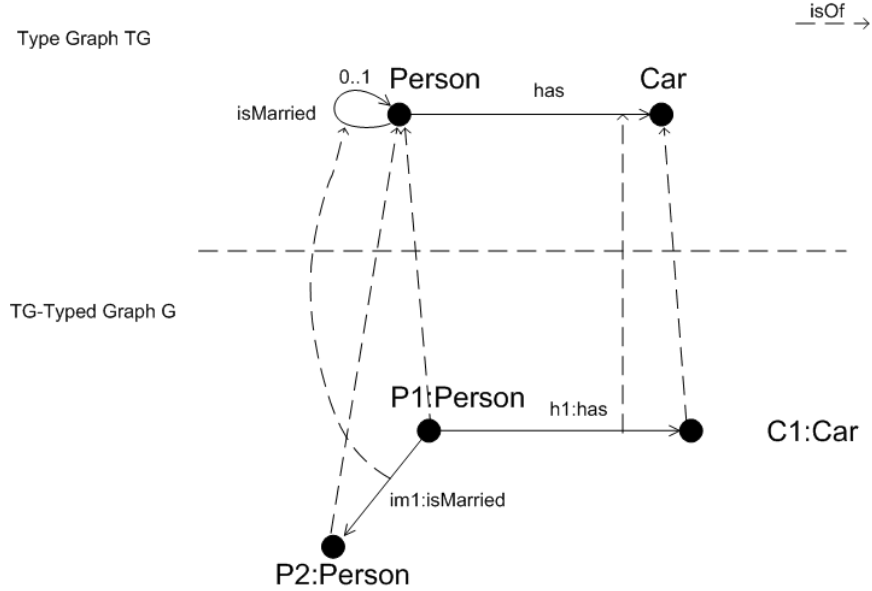


Figure C9 Example of type graph and typed graph

C2.3.2 Attributed Graph

Roughly speaking, an attributed graph is a graph in which we attach attributes to its vertices and edges. Each vertex and edge can have attached more than one attribute.

According to the definition of graph presented above, a graph is defined from a set V of vertices, a set E of Edges and two functions source and target which indicate the source vertex and the target vertex of an edge respectively. In order to define an attributed graph, we must extend this definition with:

- A set V_D of data vertices. Vertices in this set represent the values that can be assigned to attributes.
- A set E_{NA} of *vertex-attribute edges*. This set contains a type of edges whose source is a vertex of V and whose target is a data vertex of V_D . Thus, we must also introduce two new functions:
 - $source_{VA}: E_{VA} \rightarrow V$ which indicate the source of a node-attribute edge.
 - $target_{VA}: E_{VA} \rightarrow V_D$ which indicate the target of a node-attribute edge.

Appendix C

Vertex-attribute edges represent attributes of vertices. The attributes of a vertex v are those vertex-attribute edges for which the function $source_{VA}$ returns v . The type associated to each attribute is the data-vertex that is returned by the function $target_{VA}$.

- A set E_{EA} of *edge-attribute edges*. This set contains a type of edges whose source is an edge of E and whose target is a data vertex of V_D . Thus, we must also introduce two new functions:
 - $source_{EA}: E_{EA} \rightarrow E$ which indicate the source of a node-attribute edge.
 - $target_{EA}: E_{EA} \rightarrow V_D$ which indicate the target of a node-attribute edge.

Edge-attribute edges represent attributes of edges. The attributes of an edge e are those edge-attribute edges for which the function $source_{EA}$ returns e . The type associated to each attribute is the data-vertex that is returned by the function $target_{EA}$.

In order to better understand the different sets and functions that define an attributed graph we graphically illustrate them and their relations in Figure C10.

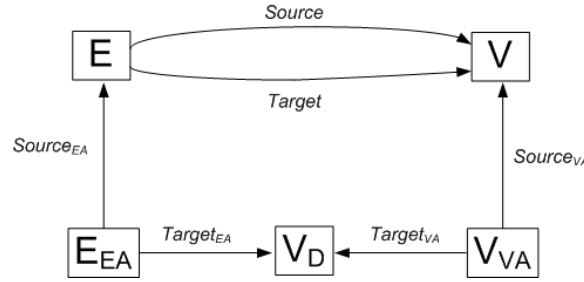


Figure C10 Sets and functions of an attributed graph

Finally, an additional aspect that must be taken into account in order to correctly define an attributed graph is that we must associate to the graph an algebra over a data signature DSIG, in the sense of algebraic signatures [Ehrig *et al.* 1985]. In this signature, we distinguish a set of attribute value sorts. These sorts are used to define the valid values that can be assigned to attributes (elements of V_D). Examples of these sorts can be:

- $D_{char} = \{a, \dots, z, A, \dots, Z, 0, \dots, 9\}$ that describe values of type char.
- $D_{string} = D_{char}^*$ that describe values of type string.

In order to see formal definitions about this aspect see [de Lara *et al.* 2005].

Appendix C

Figure C11 shows an example of attributed graph. This graph is the TG-typed graph presented in previous section which has been extended with:

1. Three data-vertices that represent the values of the attributes defined in the graph.
2. Three vertex-attribute edges. The edges *name* define two attributes for the vertex Person. Their values are “Pepe” and “Maria”. The edge *model* defines an attribute for the vertex Car. Its value is “SeatToledo”.

In order to better identify these extensions they have been depicted with dashed lines.

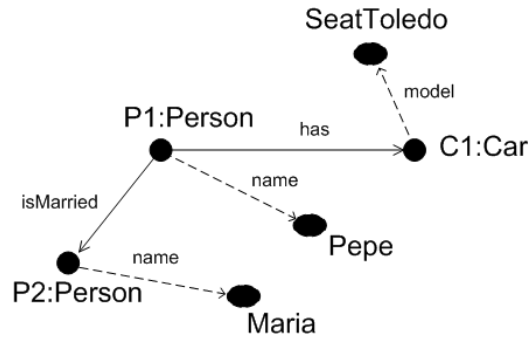


Figure C11 Example of attributed graph

C.2.3.3 Attributed Type Graphs

If we attach attributes to a type graph we are defining attributed type graphs. In these graphs, attributes are attached to types of vertices and edges. Valid attribute values are described in this case by the final algebra defined over a data signature DSIG. The use of a final algebra allows us to use sort names instead of sort elements in order to define attribute values.

Figure C12 shows an example of attributed graph. This graph is the type graph TG presented in Figure C9 that has been extended with:

1. Two data-vertices (Nat and String) which represent the types of the attributes defined in the graph.
2. Three vertex-attribute edges. The edges *name* and *age* define two attributes for the vertex person. Their types are String and Nat (Natural Number) respectively. The edge *model* defines an attribute for the vertex person. Its type is String.

In order to better identify these extensions they have been depicted with dashed lines.

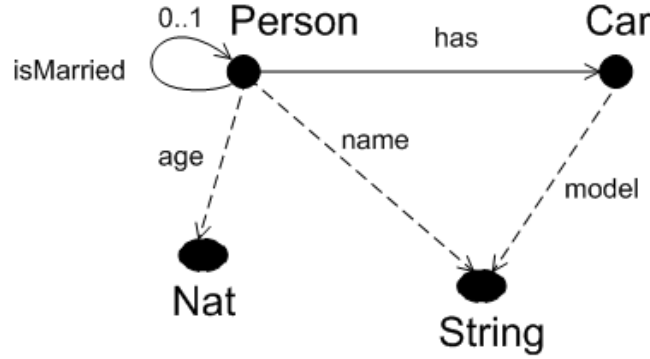


Figure C12 Example of attributed graph (an attributed type graph)

In this context, if there exists an attributed type graph ATG, we say that G is a ATG-attributed-typed graph if there are total (L, C) graph morphism of G to TG , and this morphism also preserves attributes. See [Ehrig *et al.* 2004] for a formal definition.

Finally, it is worth noting that inheritance relationships can be defined in attributed type graphs in order to solve problems related to the duplication of information (for instance, if two type of nodes share some attributes we must defined these attributes for each type of node). The inheritance in attributed type graphs are based on the use of a distinguished set of abstract nodes and inheritance relationships between the nodes. Detailed information about this topic can be found in [de Lara *et al.* 2005].

C3 Introduction to Graph Transformations

Graph transformations rely on the theory of graph grammars [Rozenberg 1997]. A graph grammar is defined from: (1) a transformation system which is a set of transformation rules and (2) a graph to which the transformation rules are applied (called *host graph*).

A graph transformation rule is defined as a set of three graphs [Löwe 1993]: LHS, RHS and K . On the one hand, LHS is the Left Hand Side of the rule and RHS is the Right Hand Side of the rule. Intuitively, the application of a graph transformation rule to a graph G is as follows: if an occurrence of LHS is found in G , it is replaced by RHS in order to obtain a derived graph H . On the other hand, the graph K , which is called the *gluing graph*, is the overlap that exists between LHS and RHS. This graph has two roles: (1) indicating which elements of the LHS are kept and (2) showing where added elements are attached during the rule application. Furthermore, two partial graph morphisms are defined: (1) one of K to LHS and (2) another of K to RHS.

Appendix C

Figure C13 shows an example of graph transformation rule. According to this rule, the edge d is deleted from the graph G . Furthermore, the vertex v_8 is added as well as the edges g and h .

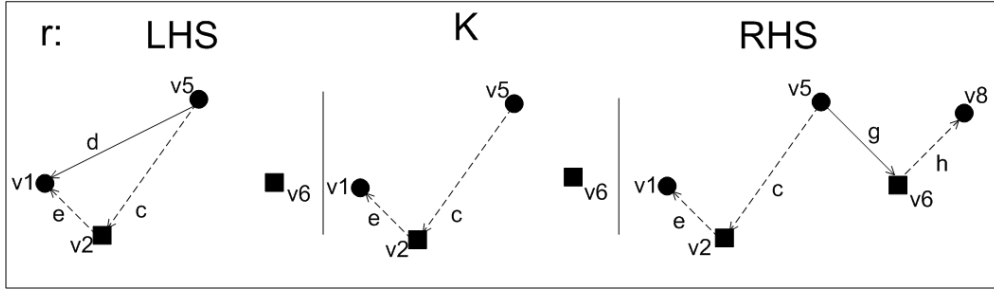


Figure C13 Example of graph transformation rule

Graph transformations are applied according to basically the following steps:

1. Find an occurrence of LHS in G . If there are several occurrences, choose one non-deterministically.
2. Remove the part of G which corresponds to $LHS - K$. Where $LHS - K$ are those elements that are in LHS and are not in K .
3. Add $RHS - K$ to G . Where $RHS - K$ are those elements that are in RHS and are not in K .

Figure C14 graphically shows how these three steps are performed when rule in Figure C13 is applied to the graph G .

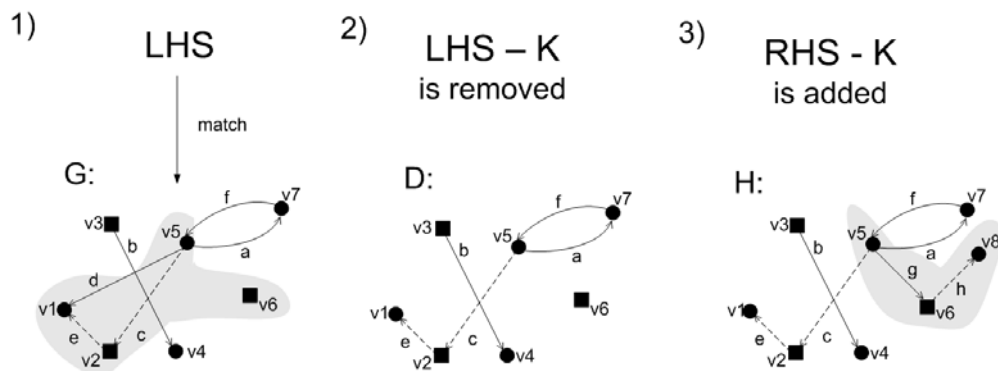


Figure C14 Example of graph transformation rule application

Appendix C

Two main algebraic transformation approaches have been introduced in the literature: the Double Pushout Approach (DPO) and the Single Pushout Approach (SPO).

Both approaches follow the three main steps presented above. The main difference between them is that DPO explicitly uses the gluing graph while SPO makes an implicit use of it. This implicit use of the gluing graph makes graph transformations to be directly defined by means of a partial graph morphism between the LHS and the RHS ($LHS \rightarrow RHS$). The rule is applied as follows: Those graph components that are related by the morphism are preserved by the rule; all the other graph components in the LHS are deleted; all the other graph components in the RHS are newly created.

Another difference between them is related to the way in which the following problem is handled: What happens when a vertex is deleted and this vertex is connected to an edge that is not considered by the rule. In this case, the result of applying the rule would no longer be a graph (there would be an edge either without source vertex or without target vertex). In the DPO approach, it is forbidden the application of rules when these situations occur. In the SPO approach, rules can be applied in these situations; however, edges connected to the deleted vertex are also deleted.

Finally, it is worth to remark that the SPO is a generalization of DPO as is stated in [Löwe 1993]. This fact makes that all the results obtained with DPO are also valid to SPO. SPO is the approach used in the context of this thesis.

C3.1 Conditional Transformation Rule Application

Conditional transformation rule application is based on the definition on pre- and post-conditions that are associated to graph transformation rules. In order to apply transformation rules these conditions must be satisfied.

There are two types of pre- and post-conditions: Positive applications Conditions (PAC) and Negative Application Conditions (NAC).

- Positive and Negative Pre-Conditions are statements that must be true and false respectively before applying the rule. Positive application pre-conditions are defined either by the LHS of a rule itself or by additional constraints specified for instance in first order logic. We can find information about this type of preconditions in [Ehrig & Habel 1986] [Pfalz & Rozenberg 1969] [Habel et al. 1996]. Negative application pre-conditions are usually defined with an additional graph structure (which must not be found in the host graph) or by other technique such as first order logic. We can find information about this type of precondition in [Ehrig & Habel 1986] [Habel et al. 1996] [Montanari 1970].

Appendix C

- Positive and Negative Post-Conditions are statements that must be true and false respectively after applying the rule. If these conditions are not verified the rule application is aborted. Information about this type of conditions can be found in [Heckel & Wagner 1995].

Finally, notice that if pre- and post-conditions are used, the algorithm to apply a rule is the following:

1. Find an occurrence of LHS in G. If there are several occurrences, chose one non-deterministically.
2. Check pre-conditions. If some pre-condition fail then abort the rule application.
3. Remove the part of G which corresponds to LHS-K. Where LHS-K are those elements that are in LHS and are not in K.
4. Add RHS-K to G. Where RHS-K are those elements that are in RHS and are not in K.
5. Check post-conditions. If some post-condition fail then cancel the modifications produced by the rule.

Appendix D

A CASE Study: An Amazon-like E-Commerce Application

D1 Introduction

In this appendix, we introduce a case study where the contributions of this thesis are put into practice. In this case study, an E-Commerce application like Amazon is developed. We have chosen an E-commerce application because this type of Web applications are focused on both providing great ammount of information to users and providing them with complex functionality in order to allow the purchase of products. These aspects allow us to properly shows the main characteristics of our approach. We have used an E-commerce application similar to Amazon in order to facilitate its understanding since Amazon is one of the most used and well-known E-Commerce applications.

The main characteristics of the case study are the follwing:

The case study is an E-commerce application that must support the on-line purchase of CD, software products and books. This E-commerce application must allow users (which are initially connected to the system as visitors) to access information about the different products that exists in the catalogue.

For each CD users must be able to know the title, the recording year, the front cover, the price some comments and the list of songs. For each software

Appendix D

product users must be able to know the name, the price, the company, an image of it, the system requirements of the software, the media in which the product software can be purchased and a brief description. For each book, users must be able to know the name, the name of the author, the price, the editorial, the year in which it was written, a summary, and the cover.

The E-commerce application must provide users with a shopping cart where they can add the products to be purchased. Users can add so many products as they want. The shopping cart must be always available to users in order to allow them to manage it. Users must be able to access the shopping cart to inquiry the products that have been added. They must also be able to remove products or modify the amount of product units that have been added .

When products are added to the shopping cart the system must automatically update the product stock, i.e. the quantity of product units that there are in the warehouse must be updated after adding the product to the shopping cart. Furthermore, the system must also register the times that a product is purchased as well as the client profiles that usually purchase this product.

Once users have selected the products to be purchased (by adding them to the shopping cart) the E-commerce application must allow them to create and send a purchase order. To do this, users must identify themselves as registered customers. To be a registered customer, they must provide the following information: its name and surname, its address, the city where they live as well as the country and the postal code, its telephone, and a login and a password. When the system asks users for identification they must introduce the login and the password.

Finally, all the information about products must be managed by Administrators. Administrators must be able to introduce new products in the catalogue, modify the information of the existent products or deleting products. Furthermore, administrators must also be able to manage the information of the registered customers. They must be able to create new customers, modify their information and delete them.

The rest of this appendix is organized as follows: Section 2 introduces the task-based requirements model that specify the requirements of the E-commerce Application. In Section 3, we show how the model-to-model transformation is applied to the task-based requirements model in order to obtain the corresponding OOWS conceptual models. Finally, section 4 shows the Web application prototype that is obtained from the OOWS conceptual models. In order to not overload this appendix each section only shows the most representative examples. A complete description of the case study can be found in [Valderas 2007a].

D2 The Task-based Requirements Model

Next, we show the task-based requirements model of the case study. First, we describe the statement of purpose. Next, we explain how task users are identified and described. Finally, we specify the data that the system must store.

D2.1 Statement of Purpose

The statement of purpose of the case of study is the following:

The Web application under development is an E-commerce Application. The main goal of the system is to provide support for the on-line purchase of products. To achieve this, the user must be able to consult information about the products, add them to a shopping cart and send purchase orders. Furthermore, tools for the management of information must be provided.

D2.2 Description of User Tasks

Next, we introduce the task taxonomy of the case of study as well as the description of elementary tasks.

D2.2.1 Task Taxonomy

The task taxonomy of the Amazon example is shown in Figure D1. It is described as follows:

- From the statement of purpose we extract the most general task that is *Purchase Products*.
- The task *Purchase Products* is decomposed by means of a structural refinement (solid line) into two tasks: (1) *Buy Products* which involves the needs related to the process of buy products and (2) *Manage Information* which involves the needs related to management activities.
- The task *Buy Products* is decomposed by means of a temporal refinement (dashed line) into *Collect Products* and *Checkout*. The relation between them is *enabling with information exchange*. Thus, the products should be collected into the shopping cart before checkout. The information that needs to be exchanged is the collected products.
- In order to collect products, users add them to the shopping cart. During this process, users can inspect the shopping cart when they consider opportune.

Appendix D

Thus, *Collect Products* is decomposed into *Add Product to Shopping Cart* and *Inspect Shopping Cart*. The relation between the two tasks is *suspend-resume*, which indicates that *Add Product to Shopping Cart* may be interrupted at any point by *Inspect Shopping Cart* (this is not mandatory, see how the second task is defined as an optional task). After each interruption, the task *Add Product to Shopping Cart* is resumed.

- *Add Product to Shopping Cart* is an iterative task. Then, it can be performed so many times the user needs. It is decomposed by means of a structural refinement into the tasks *Add CD*, *Add Software*, and *Add Book*. Then, when the user is adding a product to the shopping cart he/she is adding a CD, a Software product or a Book.
- The task *Manage Information* is decomposed by means of a structural refinement into *Manage Products* and *Manage Customers*. Users can manage information about either products or customers.
- Finally, the task *Manage Products* is decomposed by means of a structural refinement into *Manage CDs*, *Manage Software* and *Manage Books*. Users can manage information about CDs, software or books.

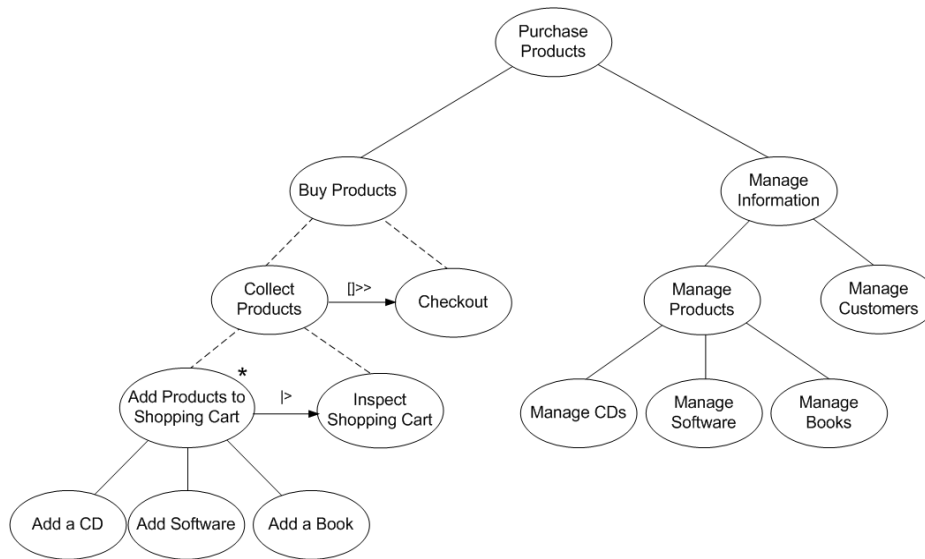


Figure D1 Task Taxonomy

D2.2.2 Description of Elementary tasks

Next, we introduce the description of some elementary tasks of the task taxonomy. In particular, we introduce the following elementary tasks: Add CD, Add Book, Inspect

Appendix D

Shopping Cart, Checkout and Manage Software. For each of these elementary tasks, we present first its characterization and next the description of the elementary task performance.

Add CD

This task describes how users must add CDs to the shopping cart. Figure D2 shows its characterization. According to this figure, the goal of the task is for the user to add a CD to the shopping cart; the task can be completed by visitors and customers. As an estimate, the task is going to be completed 100 times per hour. Finally, no additional constraints are defined.

<u>Task:</u>	Add CD
<u>Goal:</u>	The user add a CD to his/her Shopping Cart
<u>Users:</u>	Visitor, Customer
<u>Frequency:</u>	100 times per hour
<u>Additional Constraints:</u>	_____

Figure D2 Characterization of the elementary task Add CD

Figure D3 shows how the elementary task Add CD must be performed. This task starts with an *Output* IP where the system provides the user with a list (cardinality *) of music categories. From this list, the user can select a category. If the category has subcategories, the system provides (again) the user with a list of (sub) categories. If the selected category does not have subcategories the system informs about the CDs of the selected category by means of an *Output* IP.

The user can perform two actions from this last IP: (1) Select a CD, and then the system provides the user with a description of the selected CD. (2) Activate a search operation, and then the system performs a system action which searches for the CDs of an artist. To do this, the user must introduce the artist by means of an *Input* IP. If the search returns only one CD, the system provides the user with its detailed description. Otherwise, the system provides the user with a set of CDs.

Finally, when the user has obtained a CD description s/he can: (1) Select the artist of the CD, and then the system provides the user with detailed information about the artist. In this case, it is mandatory that the user returns to the CD description in order to achieve the task goal (to add a CD to the shopping cart). (2) Activate the *Add_to_Cart* operation which is an operation that allows users to manipulate IP entities (see small circle at the arc source). When users activate this operation the

Appendix D

system performs an action that adds the selected CD to the shopping cart. Next, the system updates the stock²³ and finally the task finishes.

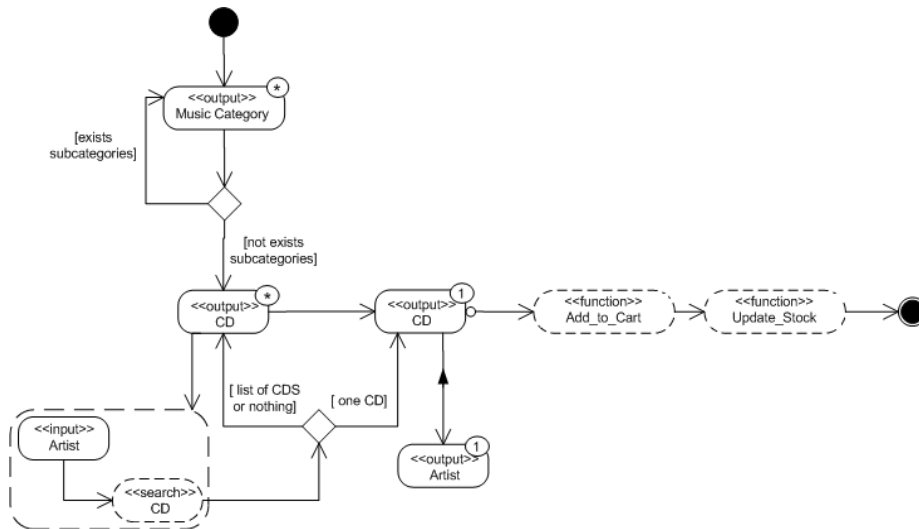


Figure D3 Description of the performance of the elementary task Add CD

Add Book

This task describes how users must add books to the shopping cart. Figure D4 shows its characterization. According to this figure, the goal of the task is for the user to add a book to the shopping cart; the task can be completed by visitors and customers. As an estimate, the task is going to be completed 100 times per hour. Finally, no additional constraints are defined.

<u>Task:</u>	Add Book
<u>Goal:</u>	The user add a book to his/her Shopping Cart
<u>Users:</u>	Visitor, Customer
<u>Frequency:</u>	100 times per hour
<u>Additional Constraints:</u>	_____

Figure D4. Characterization of the elementary task Add Book

²³ We consider that this system action also updates a list of the user profiles that usually purchase the CD as well as the times that the CD is purchased.

Appendix D

Figure D5 shows how the elementary task Add Book must be performed. The user can start this task by activating two different search system actions: (1) one that searches for books using a book property as a search criterion or (2) another that searches for books using a subject as a search criterion.

As a result of the first type of search (from a book property) a list of CDs are provided to users. As far as the second type of search (from a book subject) there are two possibilities: (1) If the subject introduced by users exists and books are found, the list of books that belongs to the subject is provided to users; otherwise, (2) users are provided with a list of subjects. From this list, users can select one and then they access the list of books that belong the selected subject.

From the list of books (accessed by any search) users can select one book and then the system provides them with a detailed description of the selected book. From this description, users can activate the *Add_to_Cart* operation which is an operation that allows users to manipulate IP entities (see small circle at the arc source). When users activate this operation, the system performs an action that adds the selected book to the shopping cart. Next, the system updates the stock and finally the task finishes.

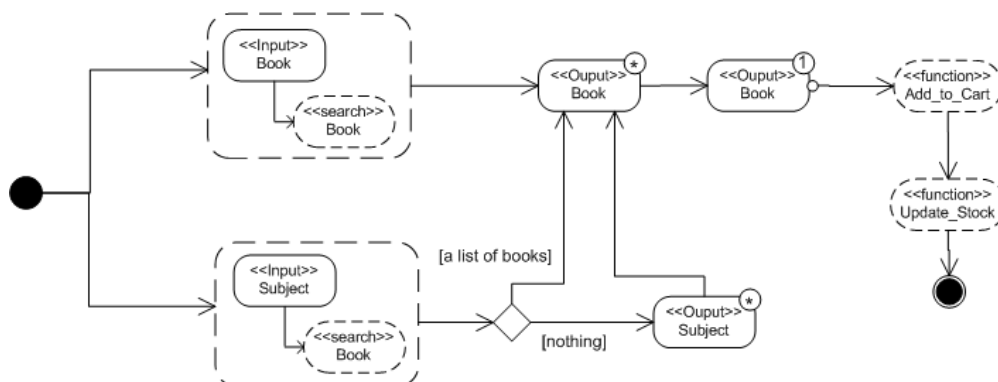


Figure D5 Description of the performance of the elementary task Add Book

Inspect Shopping Cart

This task describes how users can inspect the shopping cart. Figure D6 shows its characterization. According to this figure, the goal of the task is for the user to manage the items that have been added to the shopping cart; the task can be completed by visitors and customers. As an estimate, the task is going to be completed 40 times per hour. Finally, no additional constraints are defined.

Appendix D

<u>Task:</u>	Inspect Shopping Cart
<u>Goal:</u>	The user manage the items that have been added to his/her Shopping Cart
<u>Users:</u>	Visitor, Customer
<u>Frequency:</u>	40 times per hour
<u>Additional Constraints:</u>	_____

Figure D6 Characterization of the elementary task Inspect Shopping Cart

Figure D7 shows how the elementary task Inspect Shopping Cart must be performed. This task starts with an *Output IP* where the system provides users with a list (cardinality *) of items. For each item, users can activate two system actions: *Delete_Item* and *Modify_Item*. If the first one is activated, the system removes the corresponding item from the shopping cart. If the second one is activated, the system modifies the information associated to the corresponding item. To do this, users must introduce the new information by means of an *Input IP*.

Furthermore, users can select one item from the list and then the system provides users with a description of the product associated to the item. In this case, it is mandatory that the user returns to the list of items in order to achieve the task goal (to manage items of the shopping cart).

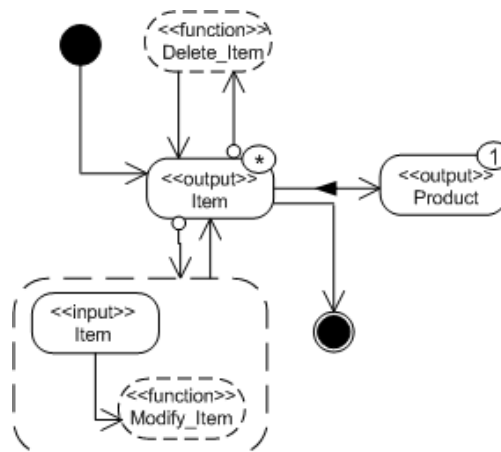


Figure D7 Description of the performance of the elementary task Inspect Shopping Cart

Appendix D

Checkout

This task describes how users can checkout once they have collected products. Figure D8 shows its characterization. According to this figure, the goal of the task is for the user to send a purchase order; the task can be completed only by customers. As an estimate, the task is going to be completed 25 times per hour. Finally, an additional constraint is defined. This constraint indicates that a security protocol must be implemented in order to perform this task.

<u>Task:</u>	Checkout
<u>Goal:</u>	The user sends the purchase order
<u>Users:</u>	Customer
<u>Frequency:</u>	25 times per hour
<u>Additional Constraints:</u>	A security protocol must be implemented to perform this task

Figure D8 Characterization of the elementary task Inspect Shopping Cart

Figure D9 shows how the elementary task Checkout must be performed. This task starts with an *Output IP* where the system provides the user with information about its purchase order. In order to follow with the task the user needs to be a registered customer. Thus, from this *Output IP*, the user can: (1) identifies itself as a register customer in order to follow with the task or (2) register itself as a customer. If the identification is successful (the user is a registered customer) the task follows by sending the purchase order. Otherwise (the user is not a registered customer) the user accesses again the *Output IP* that provides the user with the order information. From this *Output IP*, the user can register itself as a customer or try to login again.

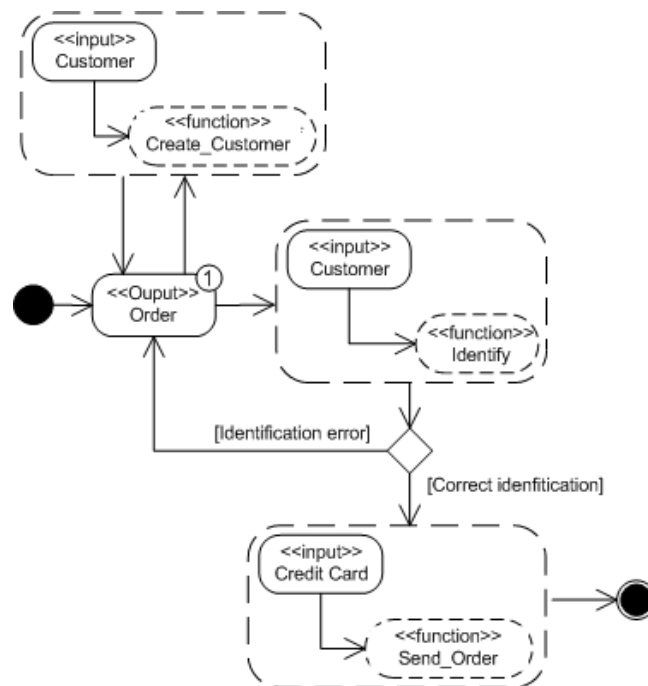


Figure D9 Description of the performance of the elementary task Checkout

D2.3 Description of the System Data

Next, we show how the information requirements are specified. We first introduce the information templates. Next, the exchanged data templates are presented.

D2.3.1 Information Templates

We define an information template for each entity identified in the task performance descriptions. As representative example we shows the following: CD, Artist, Music Category, Product, Item, and Order.

CD

The information template that is associated to the entity CD is shown in Figure D10. The information that the system must store about a CD is (see the specific data section): the CD title, the recording year, the artist that has recorded the CD, the list of songs, some comments about the CD, the front cover, the price, the number of times that the CD has been bought, the profiles of the customers which usually purchase it, the number of CD units that there are in stock and the music categories to which belong the CD. All these features present a simple nature except from the artist that has recorded the CD and the music categories.

Appendix D

Identifier:	Id1		
Entity:	CD		
Specific Data:	<i>Name</i>	<i>Description</i>	<i>Nature</i>
	Title	Title of the CD	String
	Year	Recording year of the CD	Number
	Artist	Artist that has recorded the CD	Id02
	Songs	Songs of the CD	List (String)
	Comments	A brief commentary of the CD	Text
	Front Cover	Front cover of the CD	Image
	Price	Price of the CD	Currency
	Purchase times	Times that the CD has been purchased	Number
	Client Profiles	Profiles of the clients that have purchased the CD	List (String)
	Stock	Quantity of units in stock	Number
	Music Category	Categories to which belong the CD	List (Id3)

Figure D10 Information template associated to the entity CD

Artist

The information template that is associated to the entity Artist is shown in Figure D11. The information that the system must store about an Artist is the following: the name of the artist, the nickname, the date of the artist's birth, the country where the artists was born, the record label with which the artist has signed a deal and the artist's personal web page. All these features present a simple nature.

Identifier:	Id2		
Entity:	Artist		
Specific Data:	<i>Name</i>	<i>Description</i>	<i>Nature</i>
	Name	Name of the artist	String
	Nickname	Nickname of the artist	String
	Birth Date	Date of the artist's birth	Date
	Country	Country in which the artist was born	String
	Label	Record label with which the artist has signed a deal	String
	Web Page	Web page of the artist	Url

Figure D11 Information template associated to the entity Artist

Appendix D

Music Category

The information template that is associated to the entity Music Category is shown in Figure D12. The information that the system must store about a Music Category is the following: the name of the music category, a description about it, a description of cultural aspects from which the music category is originated, a list of the instruments typically used in this kind of music and a list of subcategories. All these features present a simple nature except the last one that is defined as a list of subcategories. Each of these subcategories is described by the same information template shown in Figure D12.

Identifier:	Id3		
Entity:	Music Category		
Specific Data:	<i>Name</i>	<i>Description</i>	<i>Nature</i>
	Name	Name of Music Category	String
	Description	Description of Music Category	Text
	Cultural Origins	Description of the origins of the music category	String
	Typical Instruments	List of the instruments typically used for playing this type of music	List(String)
	Subcategories	Music subcategories derived from this one.	List (Id3)

Figure D12 Information template associated to the entity Music Category

Item

The information template that is associated to the entity Item is shown in Figure D13. The information that the system must store about an Item is the following: the product selected by the user, the quantity of product units that have been selected and the total price of the selected units. The two last features has a simple nature. The first one has a complex nature that is described by the information template in Figure D14.

Identifier:	Id8		
Entity:	Item		
Specific Data:	<i>Name</i>	<i>Description</i>	<i>Nature</i>
	Product	Product included in the item	Id9
	Quantity	Quantity of products included in the item	Number
	Total Price	Total price of the item	Currency

Figure D13 Information template associated to the entity Item

Appendix D

Product

The information template that is associated to the entity Product is shown in Figure D14. When we try to describe the specific data associated to this entity we realize that it depends on whether the product is a CD, a Book or Software. Thus, the entity Product is a super-type of the entities CD, Book and Software. In this context, depending on the product type, the specific data section associated to the entity Product is the specific data section that is associated to the entities CD, Book or Software.

Identifier:	Id9	
Entity:	Product	
Specific Data:	Super-Type Of:	
		CD: id1
		Software: id4
		Book: id5

Figure D14 Information template associated to the entity Product

Order

The information template that is associated to the entity Order is shown in Figure D15. The information that the system must store about an Order is the following: the items that are included in the order, the customer who purchase the items, the total price of the purchased items, and the credit card used to pay. Only the total prices has a simple nature. The rest of features present a complex nature (described in the information templates Id8, Id9 and Id12).

Identifier:	Id11		
Entity:	Order		
Specific Data:	<i>Name</i>	<i>Description</i>	<i>Nature</i>
	Items	List of items that are purchased	List (Id8)
	Customer	Customer that purchases the items	Id9
	Total Price	Price of the purchase	Currency
	Credit Card	Credit card used to paid the order	Id12

Figure D15 Information template associated to the entity Order

D2.3.1 Exchanged Data Templates

Next, we introduce the data exchanged templates that are defined to describe the information that is exchanged between the system and the user in each elementary task. As representative example, we present the exchanged data templates defined for the elementary tasks: Add CD and Inspect Shopping Cart.

Add CD

This elementary task presents four Output IPs and one Input IP. The exchanged data templates associated to them are presented next.

The template in Figure D16 is associated to the IP Output (Music Category,*). According to this template, the name and the description are presented for each music category that is shown in the IP.

Identifier:	O1	
IP:	Output(Music Category,*)	
Elementary Task:	Add CD	
Retrieval Condition:	-	
Exchanged Data:	Entity and Template Id	Feature
	Music Category: Id3	Name
	Music Category: Id3	Description

Figure D16 Exchanged data template for the IP Output (Music Category,*)

The template in Figure D17 is associated to the IP Output (CD,*). According to this template, the title, the price, the front cover and the artist's name are presented for each CD that is shown in the IP. Only CDs in stock are shown (see retrieval condition).

Identifier:	O2	
IP:	Output(CD,*)	
Elementary Task:	Add CD	
Retrieval Condition:	self.stock > 0	
Exchanged Data:	Entity and Template Id	Feature
	CD: id1	Title
	CD: id1	Price
	CD: id1	Front Cover
	Artist: id2	Name

Figure D17 Exchanged data template for the IP Output (CD,*)

Appendix D

The template in Figure D18 is associated to the IP Output (CD,1). According to this template, the title, the price, the front cover, the artist's name, the recording year, the list of songs, and some comments are presented for the CD that is shown in the IP.

Identifier:	O3	
IP:	Output(CD,1)	
Elementary Task:	Add CD	
Retrieval Condition:	-	
Exchanged Data:	Entity and Template Id	Feature
	CD: id1	Title
	CD: id1	Price
	CD: id1	Front Cover
	Artist: id2	Name
	CD: id1	Year
	CD: id1	Songs
	CD: id1	Comments

Figure D18 Exchanged data template for the IP Output (CD,1)

The template in Figure D19 is associated to the IP Output (Artist,1). According to this template, the artist name, the nick name, the birth date, the country, the label and the Web page are presented for the artist shown in the IP.

Identifier:	O4	
IP:	Output(Artist,1)	
Elementary Task:	Add CD	
Retrieval Condition:	-	
Exchanged Data:	Entity and Template Id	Feature
	Artist: id2	Name
	Artist: id2	Nickname
	Artist: id2	Birth Date
	Artist: id2	Country
	Artist: id2	Label
	Artist: id2	Web Page

Figure D19 Exchanged data template for the IP Output (Artist,1)

Appendix D

The template in Figure D20 is associated to the IP Input (Artist, Search (CD)). According to this template, the name of an artist is requested by the system in this IP. This data is used by a search system action in order to search CDs.

Identifier:	I1	
IP:	Input(Artist, Search(CD))	
Elementary Task:	Add CD	
Exchanged Data:	Entity and Template Id	Feature
	Artist: id2	Name

Figure D20 Exchanged data template for the IP Input (Artist, Search (CD))

Inspect Shopping Cart

This elementary task presents two Output IPs and one Input IP. The exchanged data templates associated to them are presented next.

The template in Figure D21 is associated to the IP Output (Item,*). According to this template, the quantity and the total price are presented for each item shown in the IP. Furthermore, data about the product associated to the item is also shown: the title if the product is a CD, or the name if the product is a software product or a book.

Identifier:	O10	
IP:	Output(Item,*)	
Elementary Task:	Inspect Shopping Cart	
Retrieval Condition:	-	
Exchanged Data:	Entity and Template Id	Feature
	Item: id8	Quantity
	Item: id8	Total Price
	Product: id9	As CD: Title As Software: Name As Book: Name

Figure D21 Exchanged data template for the IP Output (Item, *)

The template in Figure D22 is associated to the IP Output (Product,1). According to this template, the IP provides different information depending on the type of the products: (1) if the product is a CD, the title, the price, the year, the songs, some comments, the front cover, and the artist's name are shown; (2) if the product is a software product, the name, the price, the company, the image, the system requirements, the medium and the description are shown; and (3) if the product is a book, the name, the price, the editorial, the year, the summary, the cover and the author's name are shown.

Appendix D

Identifier:	O11	
IP:	Output(Product,1)	
Elementary Task:	Inspect Shopping Cart	
Retrieval Condition:	-	
Exchanged Data:		
As CD:	Entity and Template Id	Feature
	CD: id1	Title
	CD: id1	Price
	CD: id1	Year
	CD: id1	Songs
	CD: id1	Comments
	CD: id1	Front Cover
	Artist: id2	Name
As Software:	Entity and Template Id	Feature
	Software: id4	Name
	Software: id4	Price
	Software: id4	Company
	Software: id4	Image
	Software: id4	System Requirements
	Software: id4	Medium
	Software: id4	Description
As Book:	Entity and Template Id	Feature
	Book: id5	Name
	Book: id5	Price
	Book: id5	Editorial
	Book: id5	Year
	Book: id5	Summary
	Book: id5	Cover
	Author: id7	Name

Figure D22 Exchanged data template for the IP Output (Product, 1)

The template in Figure D23 is associated to the IP Input (Item, Modify_Item). According to this template, the quantity is requested by the system in this IP. This data is used by a function system action in order to modify the information of an item.

Identifier:	I5	
IP:	Input(Item, Modify_Item)	
Elementary Task:	Inspect Shopping Cart	
Exchanged Data:	Entity and Template Id	Feature
	Item: id8	Quantity

Figure D23 Exchanged data template for the IP Input (Item, Modify_Item)

D3 Obtaining OOWS Conceptual Models from the Task-based Requirements Model

In this section, we show the different steps that we have followed in order to obtain a skeleton of the OOWS conceptual schema from the task-based requirements model presented above. These steps, as it is explained in Chapter 6, are the following:

1. To obtain a graph representation of the task-based requirements model. To do this, we use the Task2Graph tool.
2. To transform this graph into another graph that represents an OOWS conceptual model. To do this, we use the AGG tool.
3. To translate the graph the represent the OOWS conceptual model into the proper format. To do this, we use the Graph2OOWS tool.

Next, we present the results that are obtained when we apply these steps in the development of the case of study.

D3.1 The Task-based Requirements Model as a Graph

In order to obtain a graph-based representation of the task-based requirements model presented above the tool Task2Graph (see Chapter 6) is used. Next, we present some representative examples of the obtained graphs.

Figure D24 shows the graph-based representation of both the task taxonomy and the elementary task Add CD. Figure D25 shows the graph-based representation of the elementary tasks Checkout and Inspect Shopping Cart. Figure D26 shows the graph-based representation of the information templates associated to the entities CD, Music Category and Artist. Finally, Figure D27, Figure D28 and Figure D29 show different graph representations of exchanged data templates.

Appendix D

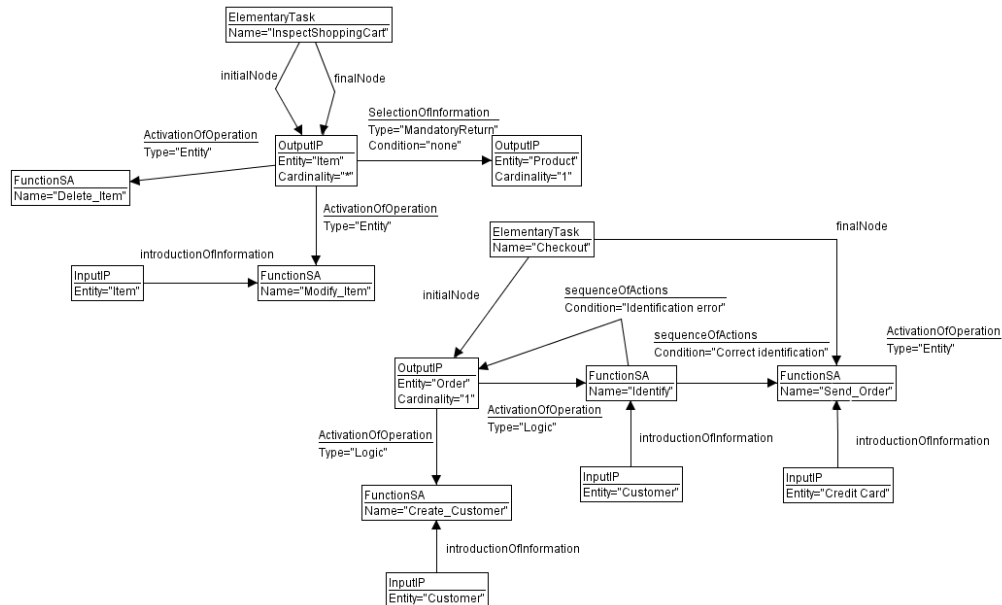


Figure D25 Graph representation for tasks Checkout and Inspect Shopping Cart

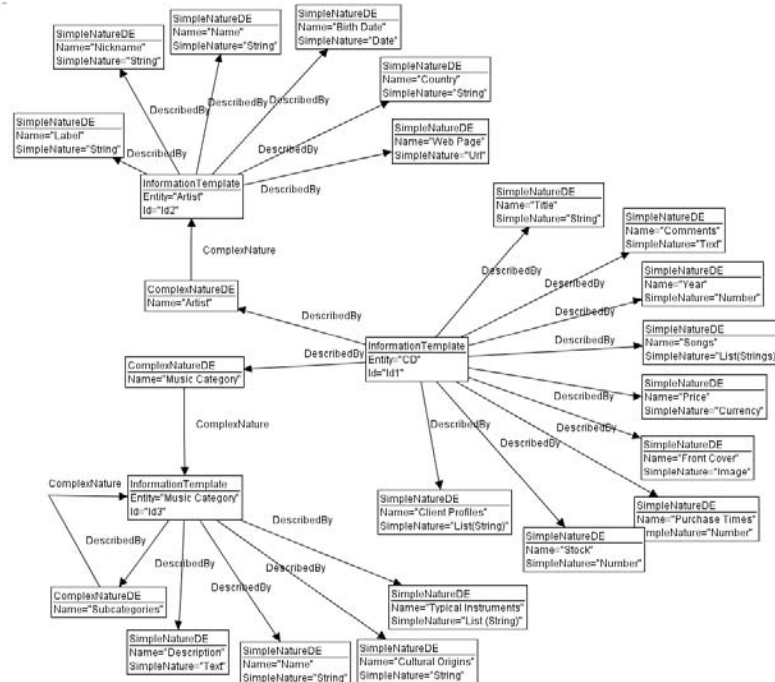


Figure D26 Graph representation for the entities CD, Music Category and Artist

Appendix D

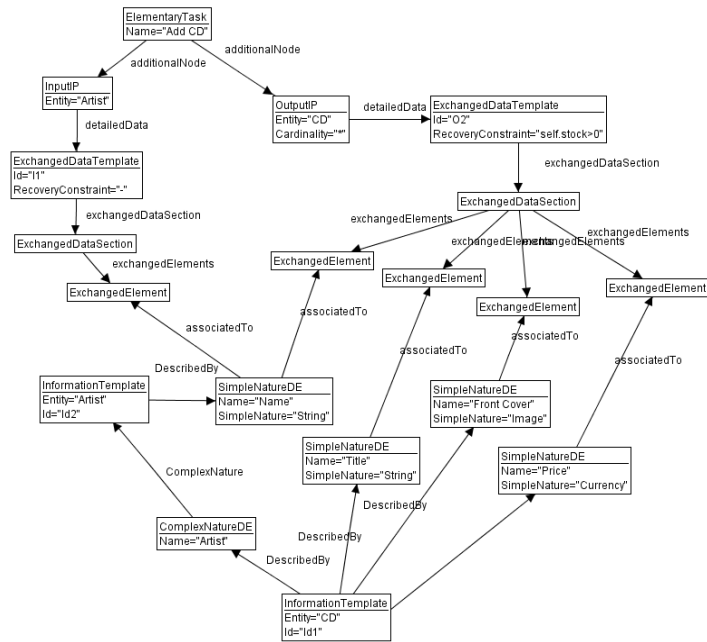


Figure D27 Graph representation for the exchanged data templates associated to the IPs Output (CD,*) and Input (Artist, Search (CD)) defined in the elementary task Add CD

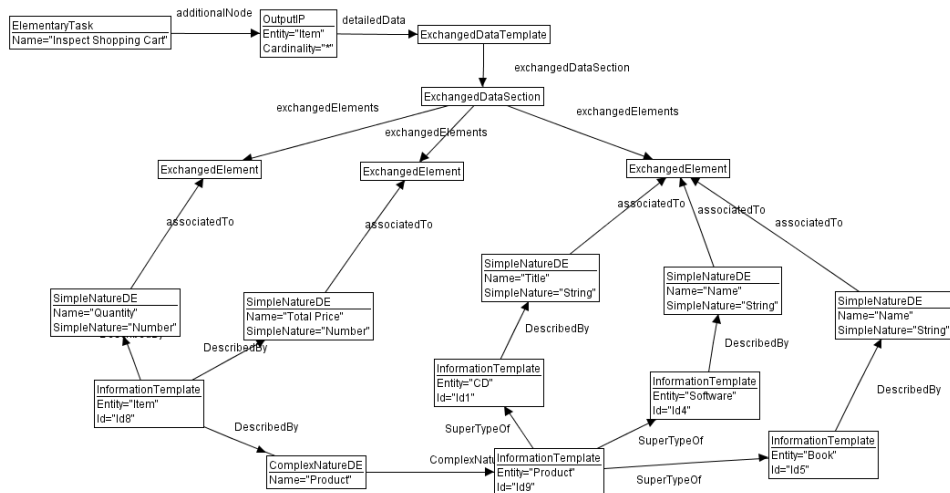


Figure D28 Graph representation for the exchanged data template associated to the IP Output (Item,*) defined in the elementary task Inspect Shopping Cart

Appendix D

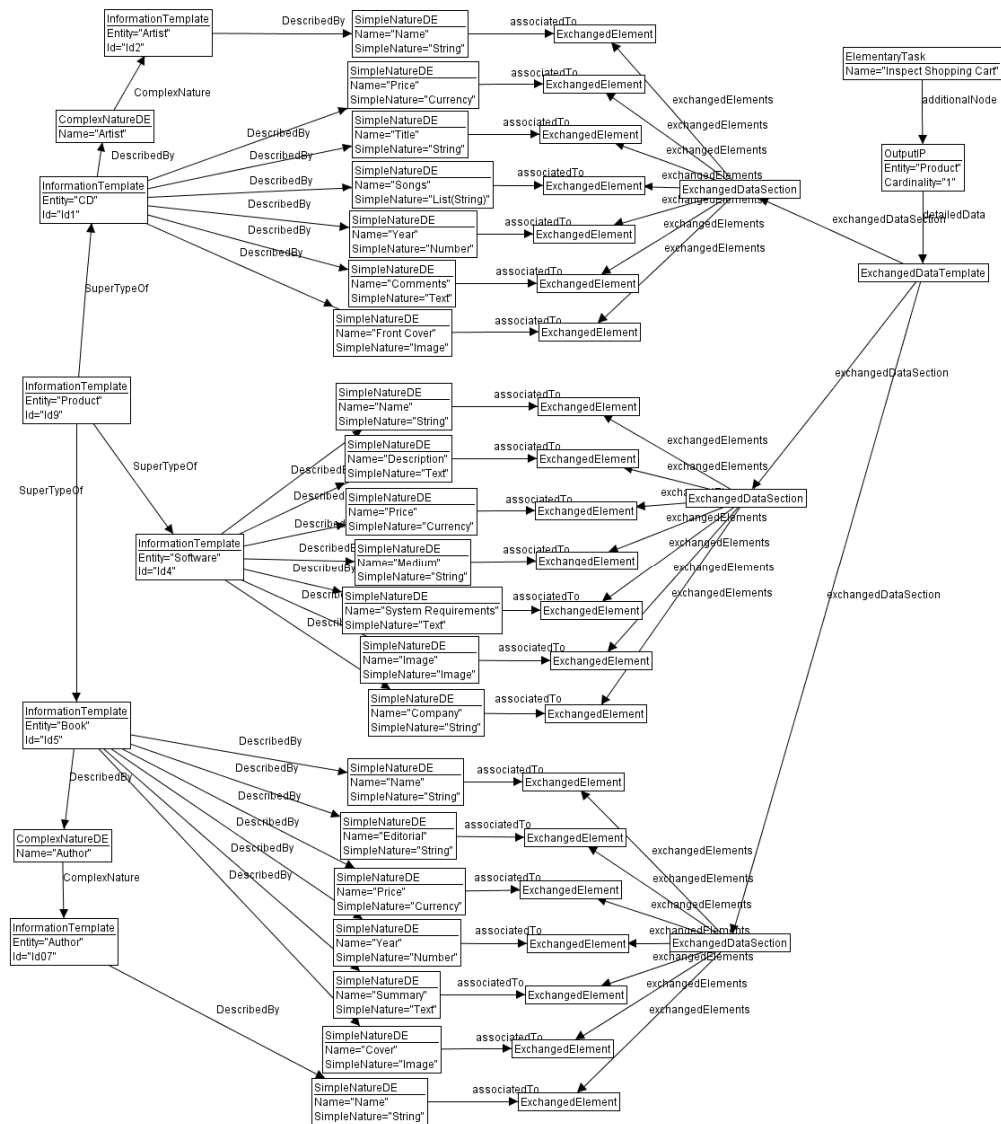


Figure D29 Graph representation for the exchanged data template associated to the IP Output (Product,1) defined in the elementary task Inspect Shopping Cart

D3.2 The Graphs Obtained after the Transformation

In this section, we introduce the two graphs that are obtained after applying the transformation by means of the AGG tool: (1) a graph that represents an OOWS structural model and (2) a graph that represents the OOWS navigational model.

D3.2.1 The OOWS Structural Model as a Graph

The graph that represents the OOWS structural model is partially shown in Figure D30 and Figure D31.

Figure D30 shows the partial graph that represents the classes CD, Music Category and Artists as well as their attributes, their operations and the relationships among them. Figure D31 shows the partial graph that represents the classes Item, Order and Product as well as their attributes and the relationships among them.

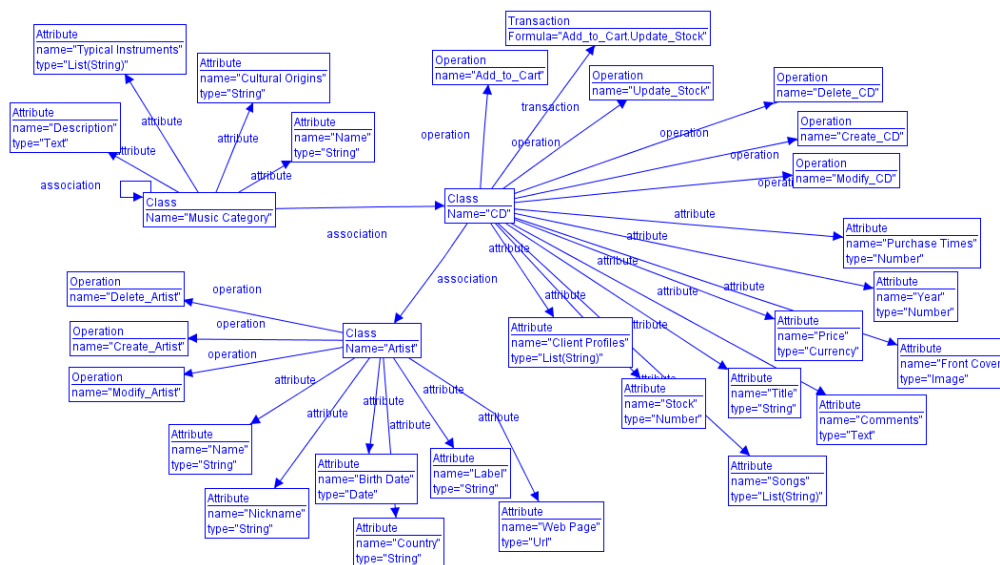


Figure D30 Graph representation of the OOWS structural model: classes CD, Music Category and Artist

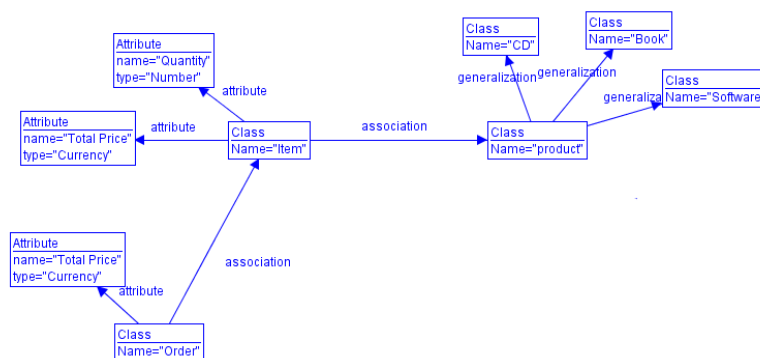


Figure D31 Graphs representation of the OOWS structural model: classes Item, Product, Order.

D3.2.2 The OOWS Navigational Model as a Graph

The graph that represents the OOWS navigational model is partially shown by the following figures:

Figure D32 shows the partial graph that represents the navigational context Music Category, CD and Artist. These navigational contexts provide support to the task Add CD.

Figure D33 shows the partial graph that represents the navigational contexts Item and Product. These navigational contexts provide support to the task Inspect Shopping Cart

Figure D34 shows the partial graph that represents the navigational context Order. This navigational context provides support to the task Checkout.

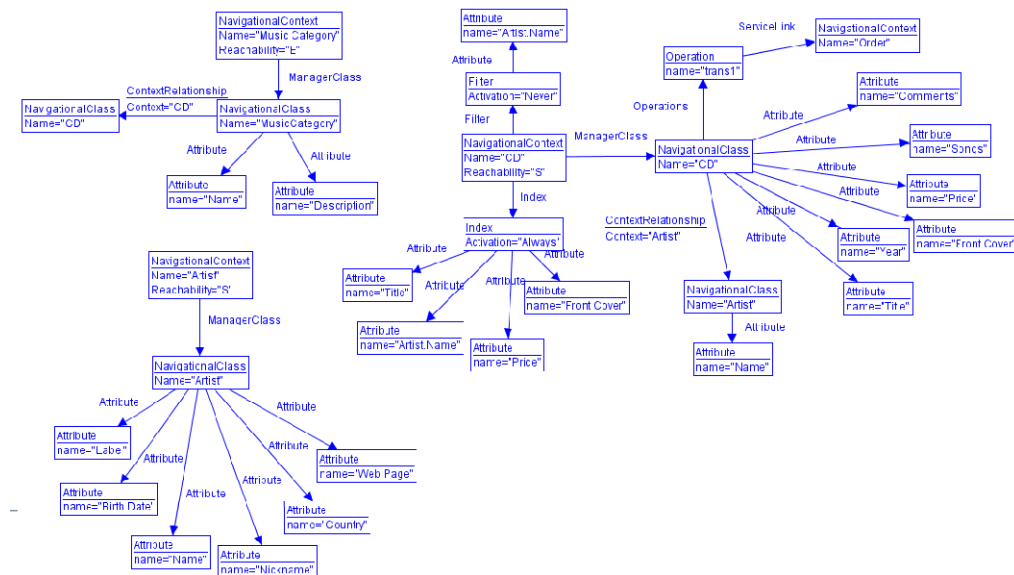


Figure D32 Graph representation for the OOWS navigational model: navigational contexts Music Category, CD and Artist.

Appendix D

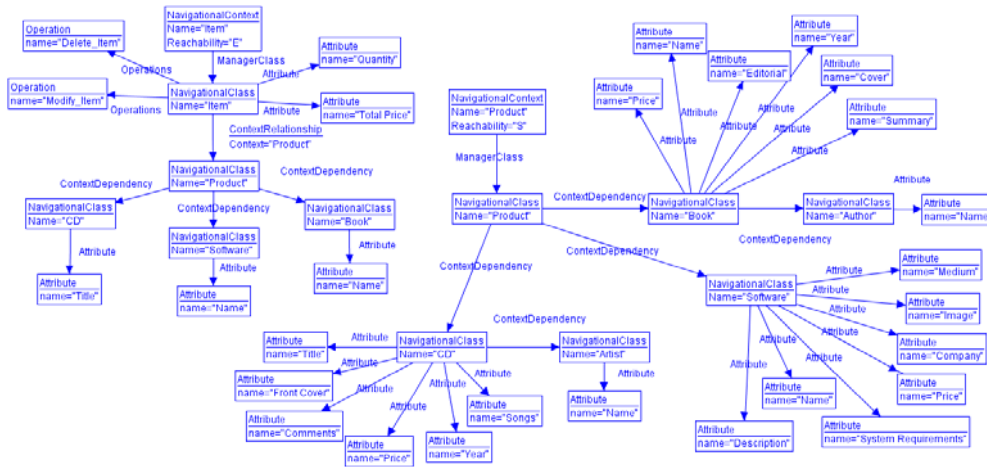


Figure D33 Graph representation for the OOWS navigational model: navigational contexts Item and Product.

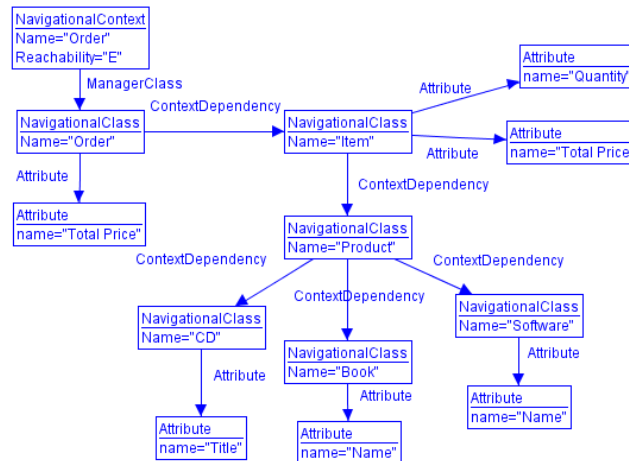


Figure D34 Graph representation for the OOWS navigational model: navigational context Order.

D3.3 The OOWS Conceptual Model

We have seen in section D3.2 the graphs that are obtained after performing the model-to-model transformations by means of the AGG tool. These graphs represent both a skeleton of the OOWS structural model and a skeleton of the OOWS navigational model. However, in order to load these skeletons in the OOWS CASE tool, we must transform them into the proper format. To do this, the Graph2OOWS

Appendix D

tool is used. We show next the final model skeletons that are obtained after using this tool.

D3.3.1 The Skeleton of the OOWS Structural Model

Figure D35 shows the skeleton of the OOWS structural model that is obtained for the case study.

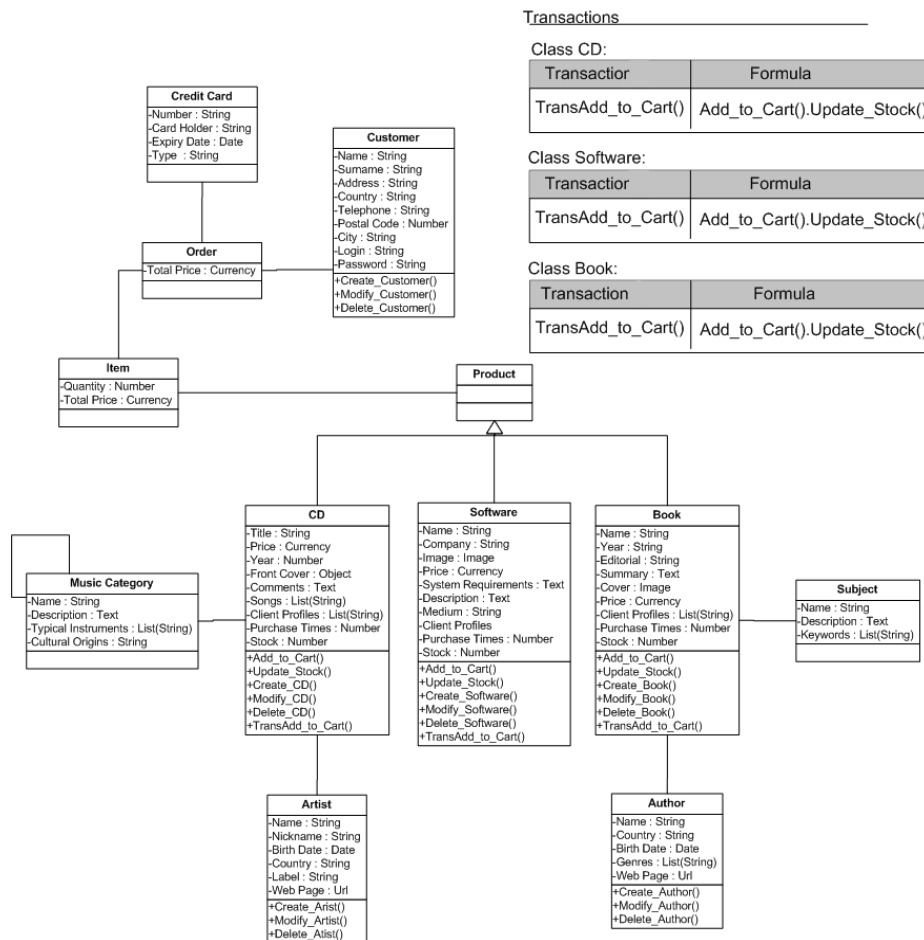


Figure D35 Skeleton of the OOWS structural model

Improving the OOWS structural model

As explained in Chapter 5, there are some aspects that cannot be systematically handled by the model transformation presented in this thesis. In the particular case of this case study, these aspects are the following:

Appendix D

- The name of each transaction is systematically defined from the string “Trans” plus the name of the first operation. It should be interesting to define names that properly adjust with transaction semantics.
- The cardinality of the different relationships is not defined.
- The operations `Add_to_Cart` and `Update_Stock` and the transaction `TransAdd_to_Cart` are shared by the classes `CD`, `Software` and `Book`. It should be interesting to generalize them to the class `Product`.

If we manually modify the OOWS structural model skeleton in order to properly support the above introduced aspects, we obtain the model presented in Figure D36.

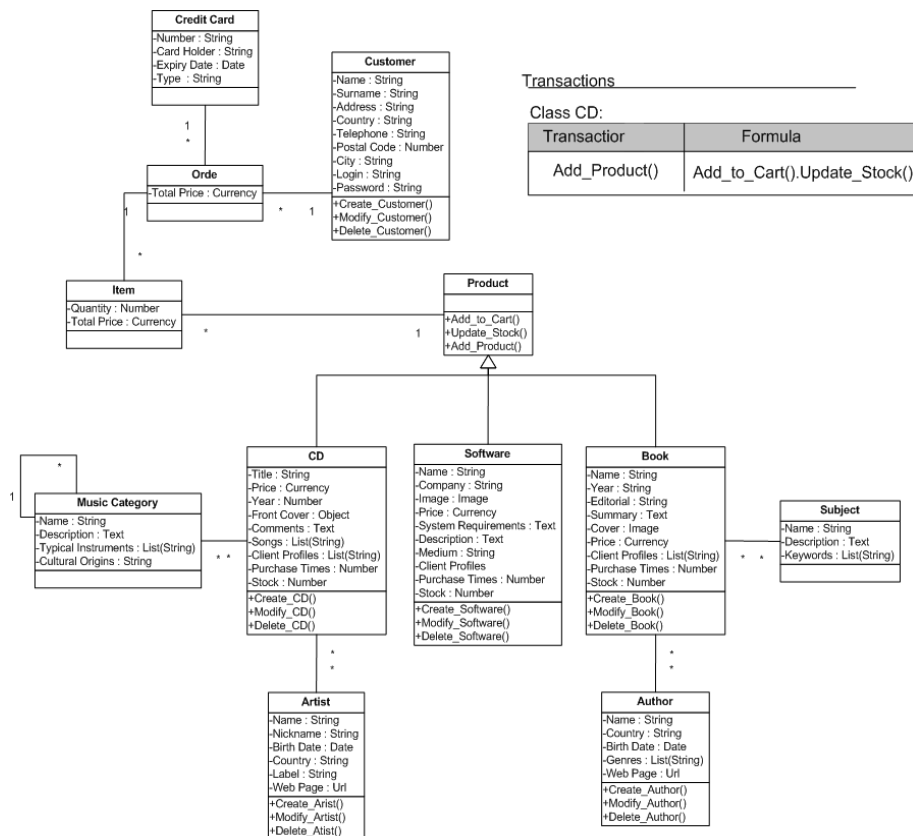


Figure D36 Skeleton of the OOWS structural model improved manually

D3.3.2 The Skeleton of the OOWS navigational model

Next, we show part of the skeleton of the OOWS navigational model that is obtained for the case study.

Figure D37 and Figure D38 show the navigational maps defined for Visitors, Customers and Administrators.

Next, we show the navigational contexts defined in the navigational map of Visitors. Figure D39 shows the navigational contexts Music Category, CD, Artist, Software, Book and Subject that provide support to perform the tasks Add CD, Add Software and Add Book. Figure D40 shows the contexts Item and Product that provide support to perform the task Inspect Shopping Cart.

Figure D41 shows the navigational context defined in the navigational map of Customers. This context is the Order context which provides support to perform the task Checkout.

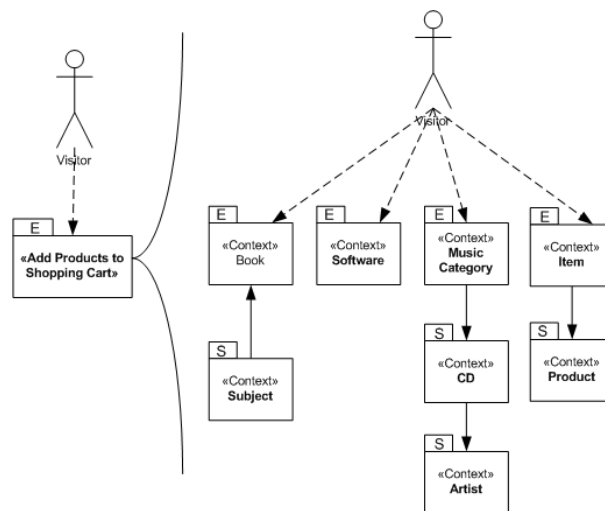


Figure D37 Skeleton of the OOWS navigational model: Visitor navigational map

Appendix D

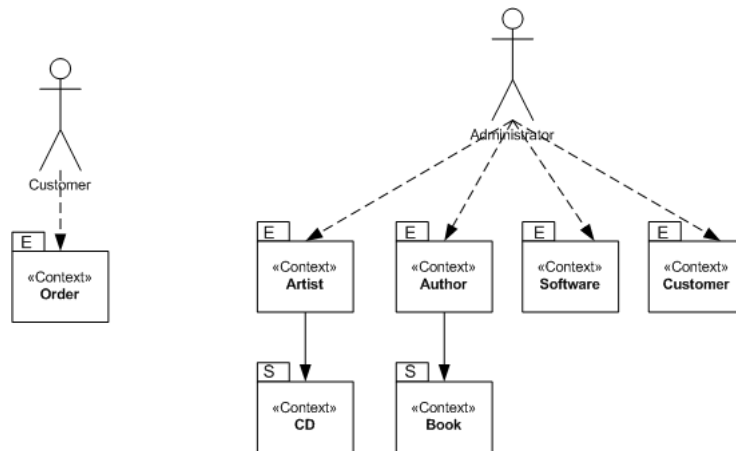


Figure D38 Skeleton of the OOWS navigational model: Customer and Administrator navigational maps

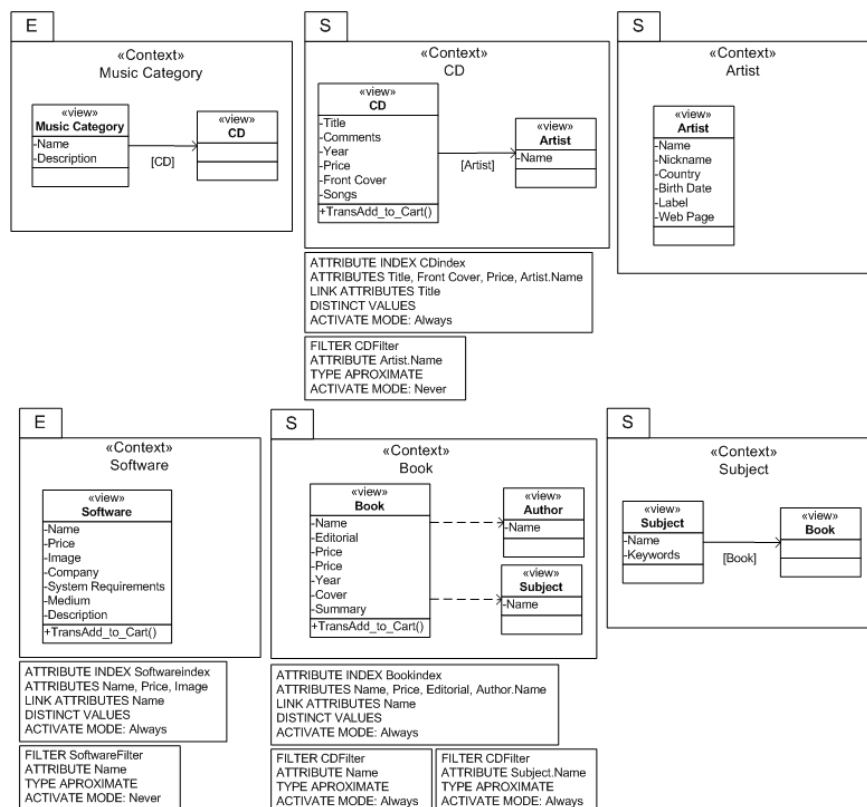


Figure D39 Contexts Music Category, CD, Artist, Software, Book and Subject for the tasks Add CD, Add Software and Add Book

Appendix D

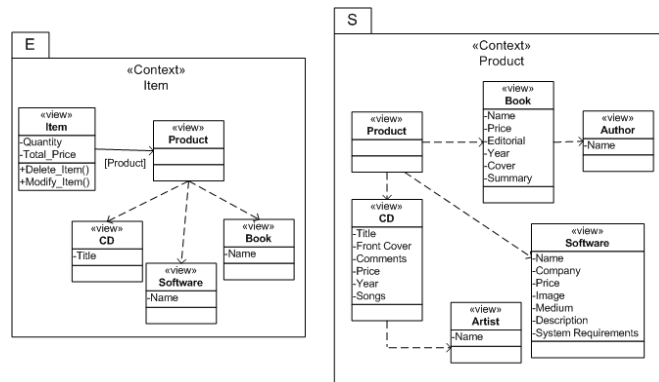


Figure D40 Contexts Item and Product for the task Inspect Shopping Cart

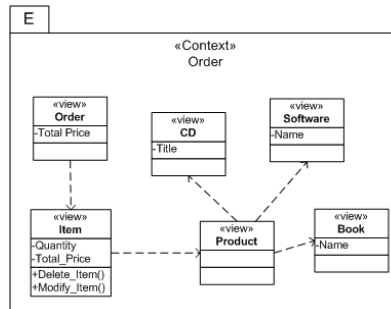


Figure D41 Context Order for the task Checkout

Improving the OOWS navigaitonal model

As explained in Chapter 5, there are some aspects that cannot be systematically handled by the model transformation presented in this thesis. In the particular case of this case study, these aspects are the following:

- **Context Names:** the names of the contexts are systematically derived when the model-to-model transformation is applied. Thus, we have manually modified them in order to better adjust to the semantics of the information and functionality provided by them.
- **Subsystems:** There is a subsystem defined in the Visitor navigational map. This subsystem is defined in order to isolate a set of navigational contexts according to the semantics of a temporal relationship defined in the task taxonomy (Suspend/Resume temporal relationship, see Chapter 5). However, after completely performing the model-to-model transformation we can see how the only navigational contexts that are defined in the Visitor navigational map are

Appendix D

those included in the subsystem. Thus, this subsystem is not required and can be removed.

- **Conditioned flows:** step flows with conditions cannot be supported by the conceptual primitives provided by the OOWS method. Most of times, this aspect produces that the obtained navigational model does not support completely the task descriptions defined in the requirements model (e.g. when a music category is selected in the Music Category context the list of CDs are provided; subcategories are not supported). Other times, this aspect produces navigational structures that need to be manually modified. For instance, the navigational context Subject defined in the Visitor navigational map should be accessed when a search filter defined in the context Book (a filter that allows users to search books by a subject name) returns nothing. However, the OOWS method does not allow us to define conditions over the results obtained by means of a search filter. In this context, there is no mechanism to access the context Subject. To solve this, we have manually added a sequence navigational link between the contexts Book and Subject.
- **Changes manually introduced in the structural model:** we have seen above how some aspects of the structural model have been manually changed in order to improve it. These changes must also be considered in the navigational model. For instance, we have changed the name of the transaction defined for adding products to the shopping cart. Then, we must also change the transaction included in the class views of the different navigational contexts.
- **Change of user role in run time:** The E-commerce application that is considered in the case study allows Visitors to access information about products and to add products to the shopping cart. However, in order to checkout, Visitors must login as registered Customers. This aspect is not systematically supported by the model-to-model transformation and we need to consider it manually. To do this, we have added an exploration link in the navigational map of the Visitor. This navigational link provides visitors with access to the context where the checkout is done (which is defined in the navigational map of Customer). However, this new exploration link is complemented with a Change of Role that forces Visitors to identify themselves as Customers in order to access the context.
- **Link attributes:** in order to define the anchor of each sequence navigational link (where users click to activate the link) the OOWS method proposes to indicate an attribute of a navigational class that is called link attribute. These attributes are not systematically derived by the model-to-model transformation. We have defined them manually.

Appendix D

Next we present an improved version of the navigational model skeleton of the case study in which the modifications introduced above has been applied manually.

Figure D43 shows the improved version of the Visitor navigational map. In this case, the subsystem has been removed and some context names have been modified in order to better illustrate the information and functionality provided by the contexts. Furthermore, the exploration link with the change of role has been included. Figure D44 shows the improved version of the Customer and Administrator navigational map. In this case, only context names are changed.

Figure D45 and Figure D46 show the navigational contexts that have been manually modified. In the navigational context Book, we have added a context navigational relationship in order to define a navigational link to the context Subject. Moreover we have modified the name of the transaction according to the changes introduced in the structural model. In the other contexts of this figure, we have only need: (1) to change the transaction name (contexts CD, and Software) or to define a link attribute (contexts Music Category, CD, Subject and Inspect Shopping Cart).

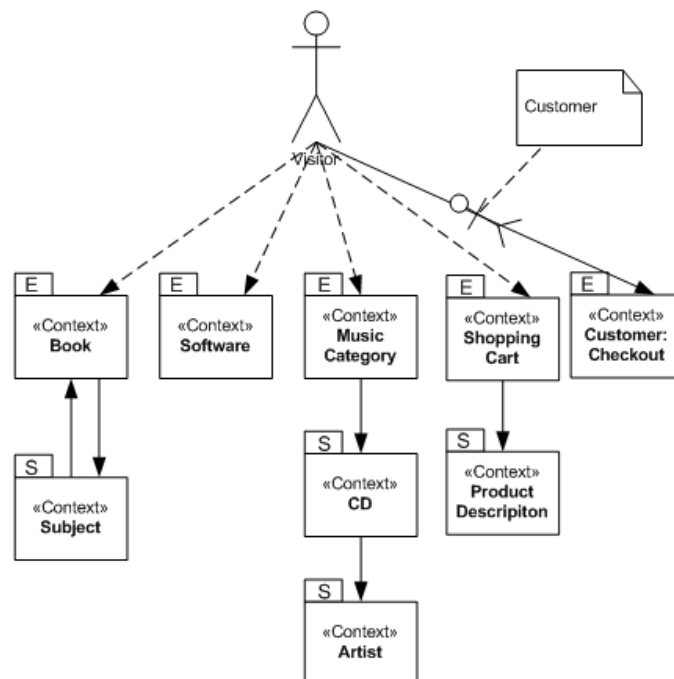


Figure D43 Skeleton of the Visitor navigational map improved manually

Appendix D

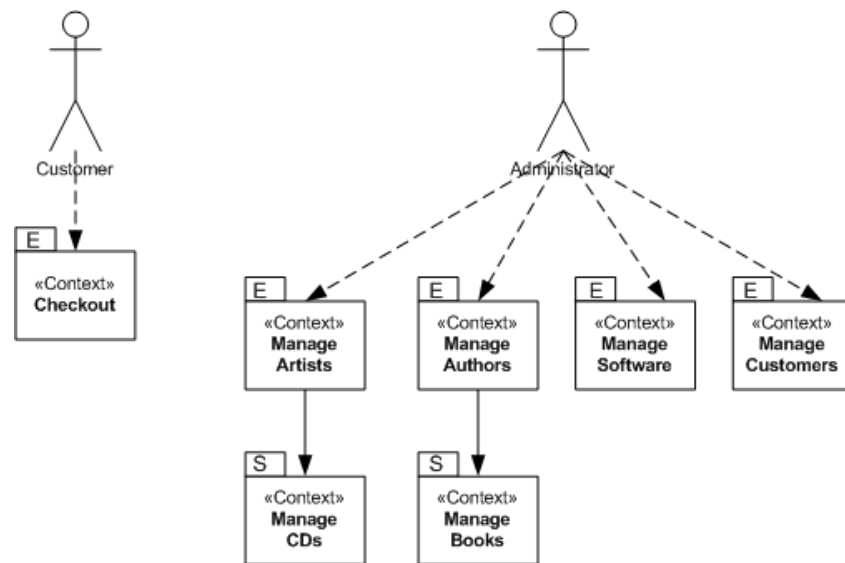


Figure D44 Skeleton of the Customer and Administrator navigational maps improved manually

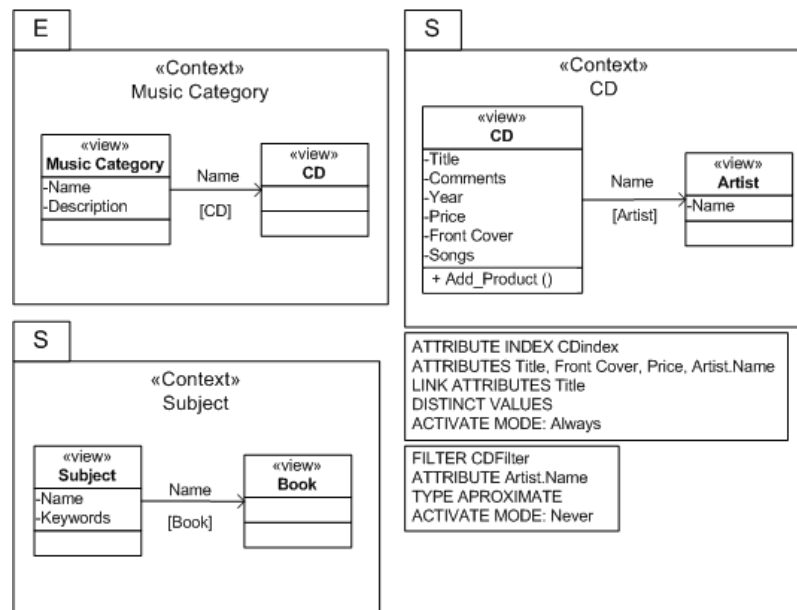


Figure D45 Navigational contexts Music Category, CD and Subject improved manually

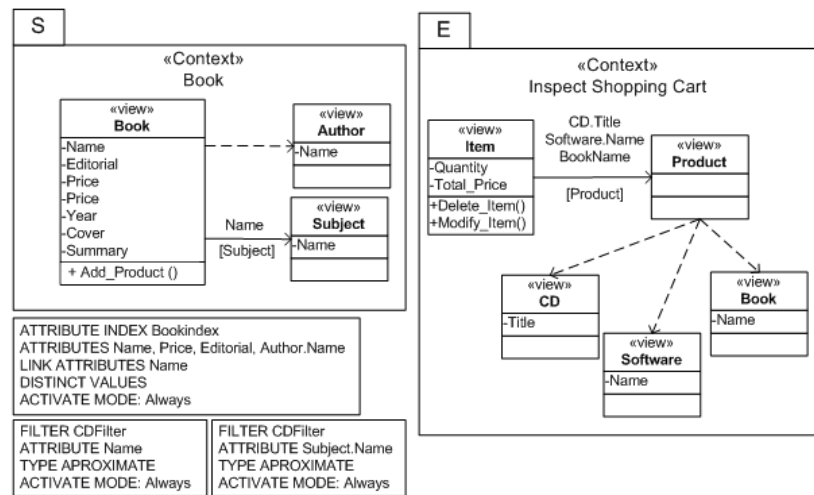


Figure D46 Navigational contexts Book and Inspect Shopping Cart improved manually

D4 Obtaining Web Application Prototypes from OOWS Conceptual Models

Throughout this appendix, we have seen the results obtained after applying the different steps of the strategy presented in Chapter 6. This strategy allows us to perform a model-to-model transformation between task-based requirements models and OOWS conceptual models. Taking the task-based requirements model of an E-commerce application as a source model (presented in Section D2), we have seen first the result obtained when the Task2Graph tool is used, which consists in a graph-based representation of the task-based requirements model (presented in Section D3.1); next we have seen how this graph is transformed into other graphs that represent OOWS conceptual models by using the AGG tool (presented in Section D3.2); and finally, we have seen the results obtained when the Grap2OOWS tool is applied, which consist in the OOWS conceptual models defined in the format that the OOWS CASE tool uses (presented in Section D3.3). Next, we present the Web application prototype that is obtained from the OOWS conceptual models presented in Section D3.3 (the improved version).

D4.1 The Obtained Web Application Prototype

In this section, we introduce the Web application prototype that is automatically obtained by applying the OOWS code generation strategy. This strategy is presented in Appendix B.

Appendix D

The Web application prototype that is generated is divided into two main folders (see Appendix B): (1) DB which contains the script for creating the database and (2) Application which contains the different files that implement the Web interface and the access to the database.

D4.1.1 The DB Folder

This folder contains a script for creating the database in which we store the information that the Web application prototype must support. The generated script is created for supporting the database server MySQL Server²⁴. Figure D47 shows a partial view of the SQL script that is obtained for creating the data base of the case study. In particular, we can see the tables created for storing information about CDs, music categories and artists as well as the tables created for storing relationships between this information.

```
create table if not exists `CD` (  
  `id` int NOT NULL,  
  `Songs` text,  
  `Front_Cover` text,  
  `Year` varchar(255),  
  `Title` varchar(255),  
  `Comments` text,  
  `Price` float,  
  `Stock` integer  
  `Purchase_Times` integer,  
  `Client_Profiles` varchar(255),  
  PRIMARY KEY (`id`)  
);  
  
create table if not exists `Music_Category` (  
  `id` int NOT NULL,  
  `Name` varchar(255),  
  `Description` text,  
  `Typical_Instruments` varchar(255),  
  `Cultural_Origins` text,  
  PRIMARY KEY (`id`)  
);  
  
create table if not exists `Artist` (  
  `id` int NOT NULL,  
  `Birth_Date` varchar(255),  
  `Genres` text,  
  `Label` varchar(255),  
  `Country` varchar(255),  
  `Nickname` varchar(255),  
  `Name` varchar(255),  
  `Web_Page` text,  
  PRIMARY KEY (`id`)  
);
```

²⁴ [Http://www.mysql.org](http://www.mysql.org)

Appendix D

```
...
create table `Music_CategoryCD` (
  `Music_Category` int NOT NULL,
  `CD` int NOT NULL,
  PRIMARY KEY (`Music_Category`,`CD`),
  FOREIGN KEY (`Music_Category`) REFERENCES `Music_Category`(`id`),
  FOREIGN KEY (`CD`) REFERENCES `CD`(`id`)
);

create table `CDArtist` (
  `CD` int NOT NULL,
  `Artist` int NOT NULL,
  PRIMARY KEY (`CD`,`Artist`),
  FOREIGN KEY (`CD`) REFERENCES `CD`(`id`),
  FOREIGN KEY (`Artist`) REFERENCES `Artist`(`id`)
);
...
```

Figure D47 Partial view of the generated SQL script

D4.1.2 The Application Folder

The Application folder contains a set of PHP files that implements the navigational structure of the Web application. This folder also contains four subfolders: (1) the *css* subfolder, in which the look and feel design of the prototype is stored; (2) the *Image* subfolder that is created for storing the different images that the Web application prototype must use; (3) the *Classes* subfolder that contains several PHP files that implement the classes defined in the structural model; (4) the *InfoQueries* subfolder that contains files that implement different functions for retrieving information from the database; (5) the *Support* subfolder that contains files for configuring the Web application prototype in run time.

Next, we introduce it in detail how the PHP files implements the navigational structure of the case study as well as the content of the subfolders *Classes* and *Support*.

The Navigational Structure

The navigational structure of the Web application prototype is implemented as a set of PHP files. These files are generated from the different navigational contexts defined in the navigational model as well as from the different access mechanisms (indexes and filters). Figure D48 shows the different files generated for supporting the Visitor navigational map.

Appendix D

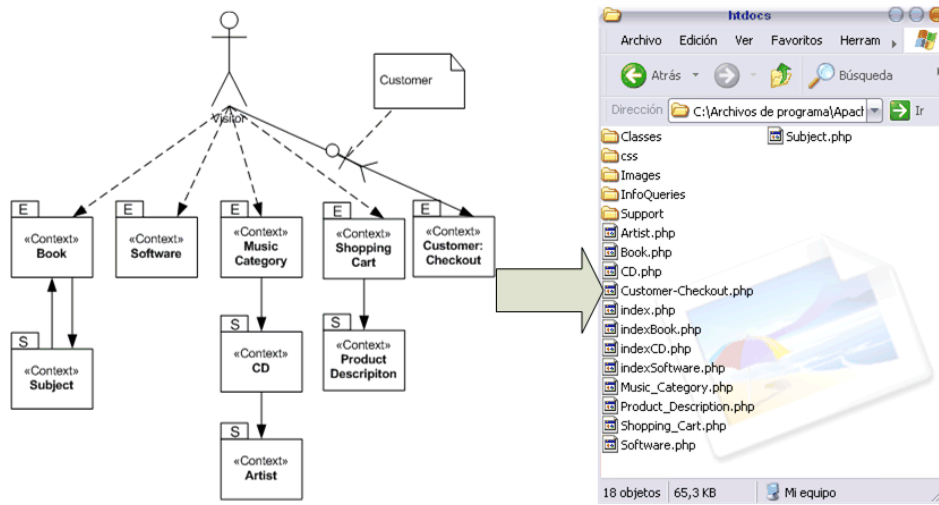


Figure D48 Generated files from the Visitor navigational map

These PHP files provide the user with the information that supports the abstract mechanisms that they implements. For instance, if a PHP file implements a navigational context it provides the information defined in the view of the navigational context; if a PHP file implements and index it provides a list defined from the attributes associated to the index. In order to retrieval this information, SQL queries are defined. A SQL query is defined for each PHP file. These queries are implemented in the file *infoquery.php*. Figure D49 shows, as representative example, the SQL query that retrieves the information of the PHP file that support the index of the context CD. This SQL query retrieves the attributes Title, Front Cover, Price and Artist Name for each CD that is in Stock. To do this, two PHP sentences are generated: one that defines the query and another to execute it. In order to execute the SQL query we use the method *consulta* that belongs to an auxiliary class defined in the file *connection_bd.php*. This file is included in the Support subfolder.

```
<? Php
require('../support/db.php');
...
function queryindexCD(){

    $consulta="select      distinct      m.id,      c0.Name,      m.Price,
m.Front_Cover, m.Title from CD m left join CDArtist dc0 on m.id=dc0.CD
left join Artist c0 on dc0.Artist=c0.id where m.Stock>0";

    $info=$db->consulta($consulta);

    Return $info;
}
?>
```

Figure D49 Generated code from retrieval data base information

Appendix D

In addition of providing users with the proper information, the PHP files included in the Application folder also provide users with the links that allow them to properly navigate the different pages. These links are created from the different exploration and sequence navigational links defined in the navigational model. Exploration links are used to define the main menu of the Web application prototype (which is usually located at the left side of each Web page). Sequence links are used to create links inside the information provided by each Web page. Figure D50 shows how the different PHP files implement the different navigational contexts and how they are interconnected according to the defined navigational links. Notice how navigational contexts with an index are implemented with two PHP files (one for the index and another for the context definition).

We can see in Figure D51 the Web page that is produced by the indexCD.php. This Web page allows users to access a list of CDs (index of the navigational context CD). In order to better visualize it a default look and feel design (defined by a set of css styles) has been incorporated. Notice moreover how the text that appears in the different options of the main menu has been automatically generated from navigational context names. We explain further how this aspect can be reconfigured by using the file *alias.php* located in the folder *Support*.

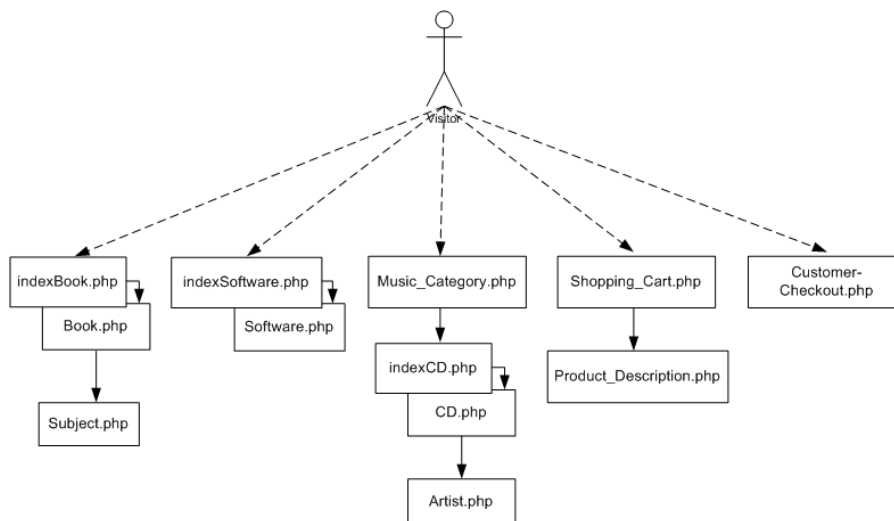


Figure D50 PHP files interconnected according to the navigational links defined in the navigational model

Appendix D

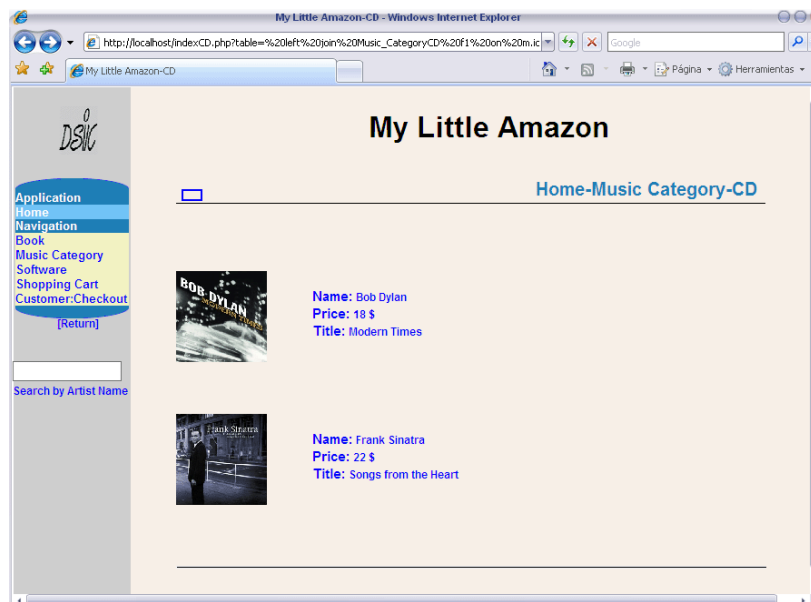


Figure D51 Example of page of the Web application prototype

The *Classes* folder

As we have explained above, this folder contains PHP files that implement the different classes defined in the structural model. These classes define the attributes of each class as well as the signatures of the methods. These classes are created in order to constitute a base for manually implementing the functionality that is not automatically generated. Figure D52 show the PHP file that implements the class CD.

```
<?php
require(' ../support/db.php' );

Class CD{

    //Attributes
    var $id int;
    var $Songs string;
    var $Front_Cover string;
    var $Year string;
    var $Title string;
    var $Comments string;
    var $Price string;
    var $Stock string;
    var $Purchase_Times string;
    var $Client_Profiles string[];

    // Default Constructor
    function CD($Songs, $Front_Cover, $Year, $Title, $Comments,
                $Price, $Stock, $Purchase_Times,Client_Profiles){
```

Appendix D

```
        //this->id= to be completed manually
        this->Songs=$Songs;
        this->Front_Cover=$Front_Cover;
        this->Year=$Year;
        this->Title=$Title;
        this->Comments=$Comments;
        this->Price=$Price;
        this->Stock=$Stock;
        this->Purchase_Times=$PurchaseTimes;
        this->Client_Profiles=$Client_Profiles;
    }

    // Operation Signatures
    createCD($Songs, $Front_Cover, $Year, $Title, $Comments,
            $Price, $Stock, $Purchase_Times, $Client_Profiles){
    }

    modifyCD($Songs, $Front_Cover, $Year, $Title, $Comments,
            $Price, $Stock, $Purchase_Times, $Client_Profiles){
    }

    deleteCD(){
    }

    add_to_Cart(){
    }

    update_stock(){
    }

    add_product(){
        $db->consulta("SET AUTOCOMMIT=0");
        $return1=this.add_to_cart();
        $return2=this.update_stock();
        if($return1==0 || $return2==0){
            $db->consulta("ROLLBACK");
        }
        else{
            $db->consulta("COMMIT");
        }
        $db->consulta("SET AUTOCOMMIT=1");
    }
}

?>
```

Figure D52 Implementation of the Class CD

The <i>Support</i> folder

The folder *Support* contains different files that provide support for configuring different aspects of the Web application prototype in run time. These files are the

Appendix D

following (see Appendix B): *bd.php* that indicates the name of the database and facilitate us to change it; *template.php* that indicate the presentation template that is being used; and *alias.php* that configures the text that appear in menus, heads, foots etc. Next, we present the content of these files for the case study.

Figure D53 shows the content of the files *bd.php* and *template.php*. These files contain global variables that create the connection to the database and establish the presentation template that is used to define the look and feel design. The rest of PHP files use these global variables in order to know this information. This aspect allows us to easily change the database and the presentation template in run time (to configure these aspects we just need to modify the value of the global variables).

<i>bd.php</i>	<i>template.php</i>
<pre><?php require('conection_bd.php'); \$db=new DataBase("dboows"); \$db->connect(); ?></pre>	<pre><?php \$template="dsic"; ?></pre>

Figure D53 Content of the files *bd.php* and *template.php*

Figure D54 shows the content of the file *alias.php*. This file defines a set of global variables that are used by the rest of PHP files in order to show the text corresponding to menu entries, headers, foots and so on.

```
<?php

$NoItems="There is no information in the Data Base";
$titleApp="My Little Amazon";
$Welcome="You can start to buy products by selecting the different
options at the left side menu";

$indexL="Home";
$Music_CategoryL="Music Category";
$BookL="Book";
$SoftwareL="Software";
$ShoppingCartL="Shopping Cart";
$CheckoutL="Customer:Checkout";

$pathT="You are here:";
$returnT="[Return]";

$CDT="CD";
$CD_Price="Price: ";
$CD_Front_Cover="Front Cover: ";
$CD_Title="Title: ";
$searchArtist_Name=" Search by Artist Name";
$CD_Songs="Songs: ";
$CD_Year="Year: ";
$CD_Comments="Comments: ";
```

Appendix D

```
$CD_Name="Name: ";  
$opAdd_to_Cart="Add to Cart";  
...  
?>
```

Figure D54 Content of the file alias.php

The variables shown above are automatically created from the navigational model. Notice how these variables establish the text that appears in the different entries of the main menu. In order to change this text we just need to modify the value of the corresponding variables, as we can see in Figure D55.

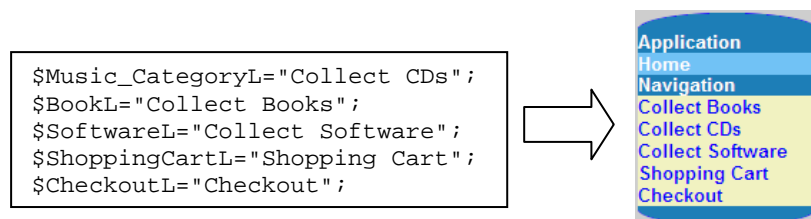


Figure D55 Modification of the main menu entries