Vrije Universiteit Brussel

FACULTY OF SCIENCES
Web & Information Systems Engineering

Error-correction for Ontology-based Websites through a Version Log

A thesis presented in fulfilment of the thesis requirement for a License Degree in Applied Computer Science by

Johan Van den Broeck

Academic Year 2005-2006

Promotor: Prof.Dr. Olga De Troyer
Supervisor: Peter Plessers

Vrije Universiteit Brussel

FACULTEIT WETENSCHAPPEN
Web & Information Systems Engineering

Foutcorrectie voor Ontologie gebaseerde Websites via een Versie Log

Eindwerk voorgelegd voor het behalen van de graad van Licentiaat in de Toegepaste Informatica door

Johan Van den Broeck

Academiejaar 2005-2006

Promotor: Prof.Dr. Olga De Troyer
Supervisie: Peter Plessers

Abstract

Ever since the creation of the WWW, one of its major advantages has also been the cause of one of its major problems: The usage of unidirectional links, which allow for the creation of links to a resource without the knowledge of the owner of the resource. The advantage of this type of links is the high degree of independence that developers can afford themselves. The problem that accompanies this advantage is the well-known problem of broken links. This is the problem that we have addressed in this thesis.

The approach that we have developed in this thesis makes use of the models of the WSDM ontology-based website design method in combination with a Version Log to provide error-correction for broken link errors. This Version Log records all the changes made to the ontology that captures the WSDM models and we use this information to be able to temporarily undo the changes made to the website that cause the broken link error.
The advantages of our approach over conventional error-correcting schemes like redirection is that our approach requires no additional input from the website engineer besides the models provided by the WSDM Method and that our approach can provide a more customized recovery, to meet the needs of the website engineer.

Abstract

Sinds de creatie van het WWW, is een van de grootste voordelen van het WWW steeds gepaard gegaan met een van de grootste nadelen: Het gebruik van unidirectionele links, die het mogelijk maken om een link te leggen naar een resource zonder dat de eigenaar van deze resource hiervan op de hoogte is. Het voordeel van dit soort links is de vrijheid die ze bieden aan de website ontwikkelaars. Het probleem dat gepaard gaat met dit voordeel is het bekende probleem van gebroken links. Dit probleem behandelen we in deze thesis.

De aanpak die we ontwikkeld hebben in deze thesis maakt gebruik van de modellen van de WSDM ontologie-gebaseerde website design methode in combinatie met een Versie Log om fouten van het gebroken link type te corrigeren. Deze Versie Log registreert al de wijzigingen die gedaan worden aan de ontologie die de WSDM modellen omvat en we gebruiken deze informatie om de wijzigingen die de oorzaak zijn van de gebroken link fouten tijdelijk ongedaan te maken.
De voordelen van onze aanpak ten opzichte van conventionele fout-corrigerende methodes zoals redirectie, zijn, dat onze aanpak geen extra input vereist van de website ontwikkelaar naast de WSDM modellen en dat onze aanpak de mogelijkheid biedt voor een fout-correctie die beter aangepast is aan de noden van de website ontwikkelaar.

Contents

1 Introduction **1**
1.1 Motivations . . . 1
1.2 Structure of this thesis . . . 4

I Background **5**

2 Ontologies **6**
2.1 Definition . . . 6
2.2 Web Ontology Languages . . . 7
2.2.1 RDF-S . . . 8
2.2.2 OWL . . . 11

3 Ontology Based Website Design Methods **15**
3.1 WSDM . . . 16
3.1.1 Mission Statement Specification . . . 17
3.1.2 Audience Modeling . . . 17
3.1.3 Conceptual Design . . . 17
3.1.4 Implementation Design . . . 20
3.1.5 Implementation . . . 21
3.1.6 Adaptation . . . 23
3.2 Hera . . . 24
3.2.1 Conceptual Design . . . 25
3.2.2 Application Design . . . 27
3.2.3 Presentational Design . . . 30
3.3 OntoWeaver . . . 32
3.3.1 Domain Ontology Design . . . 33
3.3.2 Navigational Structure Specification and User Interface Composition . . . 33
3.3.3 Layout and Presentation Style Definition . . . 35
3.3.4 Customization Requirements Specification . . . 36

3.4 SHDM 38
3.4.1 Conceptual Design 40
3.4.2 Navigational Design 40
3.4.3 Interface Design 42
3.5 Conclusion 44

4 Ontology Evolution **45**
4.1 SIKS Ontology Evolution Approach 45
4.1.1 Change Representation 46
4.1.2 Change Detection 48
4.1.3 The Change Process 49
4.2 KAON Ontology Evolution Approach 51
4.2.1 Change Representation 52
4.2.2 Semantics of Change 53
4.2.3 Change Propagation 54
4.2.4 Change Implementation 55
4.3 WISE Ontology Evolution Approach 58
4.3.1 Change Request 59
4.3.2 Consistency Maintenance 59
4.3.3 Change Detection 60
4.3.4 Change Recovery 61
4.3.5 Change Implementation 61
4.3.6 Cost of Evolution 61
4.3.7 Version Consistency 62
4.3.8 Version Ontology 62
4.3.9 Change Definition Language 63
4.4 Conclusion 63

5 WSDM Ontology **65**
5.1 Object Chunks 65
5.2 Navigational Model 67
5.3 Site Structure Model 69
5.4 Presentation Modeling Concepts 69
5.4.1 Primitive Presentation Concepts 69
5.4.2 Complex Presentation Concepts 70
5.5 Behavior Modeling Concepts 71
5.6 Style & Template Model 72
5.7 Page Model 73

II Research 74

6 Changes 75
6.1 Possible Changes . . . 76
6.2 Unsupported Changes . . . 78
6.2.1 Inapplicable Changes . . . 78
6.2.2 Changes Beyond The Scope . . . 79
6.3 Supported Changes . . . 80
6.3.1 Changes Captured By WSDM . . . 80
6.3.2 Changes Captured By Our Approach . . . 81

7 Supporting Changes 83
7.1 Goals . . . 83
7.2 Solution Overview . . . 83
7.2.1 Architecture . . . 84
7.2.2 Components . . . 85
7.2.3 Process . . . 89
7.3 Detailed Solution . . . 90
7.3.1 Component Extensions . . . 90
7.3.2 Detailed Process . . . 98

8 Illustrative Example 119
8.1 Example Website . . . 119
8.1.1 The IM . . . 119
8.1.2 The Version Log . . . 123
8.1.3 The Implementation . . . 123
8.2 Example Changes . . . 126
8.2.1 Change 1: Renaming . . . 126
8.2.2 Change 2: Removing . . . 127
8.3 Example Approach Application . . . 131
8.3.1 Step 1: Check the existence . . . 131
8.3.2 Step 2: Retrieving the website versions . . . 131
8.3.3 Step 3: Finding the link . . . 132
8.3.4 Step 4: Find Renamed . . . 134
8.3.5 Step 5: Retrieve link status . . . 135
8.3.6 Step 6: Find removed . . . 135
8.3.7 Step 7: Return the snapshot resource request . . . 136
8.3.8 Step 8: Retrieve concepts . . . 136
8.3.9 Step 9: Transform Concepts . . . 136
8.3.10 Step 10 & 11: Retrieve website versions and Find Link . . 137
8.3.11 Step 12: Find link version . . . 137

8.3.12 Step 13: Return the live-time resource request 137

9 Proof Of Concept **138**
9.1 Versioning 139
9.1.1 Change 1: Renaming 140
9.1.2 Change 2: Removing 141
9.2 Recovering 142
9.2.1 Website Versioning 143
9.2.2 Resolving a URL 145
9.2.3 Recovering the IM 148

10 Conclusion **151**

11 Acknowledgements **153**

10 References **153**

List of Figures

2.1 Semantic Web Layers 8
2.2 Vehicle Class 9

3.1 WSDM Phases 16
3.2 WSDM Task Model 18
3.3 WSDM Object Chunk 19
3.4 WSDM Subtask Navigational Model (1) 20
3.5 WSDM Subtask Navigational Model (2) 21
3.6 WSDM Architecture 22
3.7 Hera WIS architecture 24
3.8 Hera Conceptual Model 26
3.9 Hera Navigation Data Model 27
3.10 Hera Application Model 28
3.11 Hera Application Model Adaptation 29
3.12 Hera Extended Application Model 30
3.13 Hera Presentation Model 31
3.14 Ontoweaver Framework Overview 32
3.15 Ontoweaver Domain Model 33
3.16 Ontoweaver Site Structure Model 34
3.17 Ontoweaver Site View Ontology 35
3.18 Ontoweaver Presentation Ontology 36
3.19 Ontoweaver Customization Framework 37
3.20 SHDM Conceptual Model 40
3.21 SHDM Navigational Class Model 41
3.22 SHDM Navigational Context Model 42
3.23 SHDM Context Definition Card 43
3.24 SHDM Abstract Interface 43

4.1 SIKS Change Representations 46
4.2 SIKS Ontology Change Process Model 50
4.3 KAON Ontology Evolution Process 51

4.4 KAON Ontology Changes Abstraction Layers 52
4.5 KAON Evolution Ontology and Evolution Log dependencies . . . 56
4.6 KAON Evolution Ontology 57
4.7 WISE Ontology Evolution Framework Phases 58
4.8 WISE Ontology Evolution Framework Version Consistency . . . 62
4.9 WISE Ontology Evolution Approach Version Ontology 63

5.1 WSDM Information & Functional Modeling Concepts 66
5.2 WSDM Object Chunk Functions 67
5.3 WSDM Navigational Model 67
5.4 WSDM Primitive Presentation Model Concepts 70
5.5 WSDM Complex Presentation Model Concepts 71
5.6 WSDM Behavior Model Concepts 71
5.7 WSDM Template Modeling Concepts 72

7.1 Error-correction Architecture 84
7.2 WSDM Page Links 85
7.3 WSDM Transformation Pipeline 87
7.4 Thesis Link Temporal Context Example 91
7.5 Thesis Link Temporal Context Extension 93
7.6 Thesis Deletion Extension 94
7.7 Thesis WebSite Extension 95
7.8 Thesis Link Filename Extension 96
7.9 Thesis Detailed Livetime Process 99
7.10 Thesis Detailed Snapshot Process 100

8.1 Thesis Example IM 120
8.2 Thesis Example MemberDescriptor Object Chunk 121
8.3 Thesis Example Link Instance 122
8.4 Thesis Example URL 122
8.5 Thesis Example Link Version Log 124
8.6 Thesis Example Member Page 125
8.7 Thesis Example Index Page 125
8.8 Thesis Example Changed Link Instance 127
8.9 Thesis Example Changed Link Version Log (1) 128
8.10 Thesis Example IM Changed 129
8.11 Thesis Example Changed Link Version Log (2) 130
8.12 Thesis Example Link Version Log Query 133

9.1 Thesis Implementation Protege Ontology Editor 139
9.2 Thesis Implementation Version Log Plugin Intial Version 140
9.3 Thesis Implementation Renaming Change IM 141

9.4 Thesis Implementation Renaming Change Version Log 142
9.5 Thesis Implementation Removing Change Version Log 143
9.6 Thesis Implementation Website Versioning 144
9.7 Thesis Implementation Website Versioning IM 145
9.8 Thesis Implementation Website Versioning Version Log 146
9.9 Thesis Implementation URL Checking 147
9.10 Thesis Implementation URL Checking Result 148
9.11 Thesis Implementation IM Rebuilding 149
9.12 Thesis Implementation Transformed URL 150

List of Tables

2.1 OWL sub languages 11
3.1 SHDM Artefacts 39

Chapter 1

Introduction

1.1 Motivations

In 1990, the Proposal for the *World Wide Web (WWW)* [1] was published by Berners-Lee, laying the foundations for the current WWW. Berners-Lee combined the existing *hypertext* [2] and *internet* technologies and while doing so developed the *Uniform Resource Identifier* (URI) [3] system, a system of global unique identifiers on the Web. The main differences of the WWW with other hypertext systems available at the time were the following[1]:

- The WWW was non-proprietary, allowing the independent development of servers, clients and extensions.
- The WWW uses unidirectional instead of bidirectional links, allowing links to a resource without requiring action of the resource's owner. This decentralized approach allowed further independence for developers, but also created the problem of broken links. This problem will be addressed in this thesis.

The real breakthrough for the WWW came in 1993, with the creation of Mosiac[2], the first graphical web browser. Soon, the WWW became the most popular Web application in a time when access to the internet was spreading beyond members of the academic and research community. In the years following its breakthrough to the general public, the WWW has become a large data-centric network of web applications.

[1]World Wide Web, Wikipedia, see `http://en.wikipedia.org/wiki/World_Wide_Web`

[2]Mosiac (web browser) Wikipedia, see `http://en.wikipedia.org/wiki/Mosaic_web_browser`

As mentioned before, one of the WWW's properties that led to it's success was it's decentralized approach to resource linking. This approach, where links to resources can be created without knowledge of the owner of the resource can lead to major problems:

- The contents of the resource can change, removing the information *identified* by the link. This leads to a link pointing to an existing resource containing the wrong information, without the knowledge of the owner of the resource.
- The resource itself can be removed by the owner, intentional or not. The result is a link pointing to a lost resource.
- The resource can be moved to another location, due to a restructuring of the website. The result is a link pointing to a missing resource, but this resource is still available from another location on the same website.

These problems corrupt the concept of an URI, as used by a hypertext link[3], a fundamental component of the WWW. The URI (”Uniform Resource Identifier”) no longer ”identifies” the resource in question.
Studies [7] have found the half-life of Web Resources to range from 2 to 4 years depending on the type of resource. With resource type meaning the origins of the resource (e.g., random web page, a digital library object, a legal citation or a computer science citation) and half-life meaning the time it takes for half of a defined amount of resources to disappear.

A number of practices have been devised to address these problems:

Server Redirects Server redirects simply work by sending the address of the new location of the resource to the browser, which then sends the new location back to the server. Most modern browsers will support this, but other user agents, like indexing and auditing tools may not and so provide incorrect information [8].

Similarity Retrieval In this case, the server attempts to find existing resources similar to a cached version of the missing resource and redirect the browser to the found resource. Although additional efforts have been made to improve upon this [9], the accuracy of this method still depends on the content similarity of the resource. As this method also uses server redirects, certain user agents will have trouble handling these redirects.

[3]Cool URIs don't change, see `http://www.w3.org/Provider/Style/URI.html`

Mirrors Mirrors may be useful for maintaining resources that were accidentally deleted, but provide no solution when the resource has been moved deliberately. Besides the problems arising from creating a static mirror of dynamic resources [5], web site administrators will again have to make use of a server redirection scheme and the detection problems it brings along.

All too often, however, no attempt is made to solve the problem and the user is simply presented with a so-called *404* error page [6]. These error pages are seldom helpful and novice users will simply give up on the resource [4].

Our approach for solving these problems, which we describe in this thesis, needs structural information, this structural information can be obtained from *structured* website design methods

Ever since the creation of the WWW, web sites have become increasingly complex. Because of this increased complexity, ad-hoc design methods are no longer sufficient since they bring along several usability problems [10]:

- Redundancy: Needlessly repeated information during navigation is annoying to users.
- Inconsistency: If information on the website is inconsistent, the user will probably distrust the whole website.
- Incompleteness: This mean stale and broken links, but also lack of information that users expect to be available on a site.
- Actuality. If a website has visibly not been updated for a while, the users will most likely not trust the provided information.

A number of structured *design methods* have been developed to overcome these problems. These design methods, like WSDM [14], Hera [16], OntoWeaver [17] and SHDM [32] make use of models to specify websites at a *conceptual* level. *Ontologies* [11] are used to formally capture these models and add extra high-level structural and content annotations to websites.

In this thesis, Ontology Evolution [13] will be used as an innovative way to track changes to the ontologies that make up the higher-level structure of the website. The information acquired by the tracking of these changes will then be used to correct errors caused by moved or missing resources. The error-correction achieved in this manner is both automated and effective.

1.2 Structure of this thesis

This thesis is structured as follows: there are two parts, the first part covers the background needed to understand this thesis and the second part covers the research done. The research part starts by giving an overview of the error-correction approach, followed by a specification of its different components.

Background Chapter 2 gives information about ontologies. Chapter 3 gives an overview of the most prominent ontology based website design methods. Chapter 4 discusses ontology evolution. And Chapter 5 describes the ontologies used by the WSDM design method.

Research In Chapter 6 we define the scope of the problems that will be handled by the error-correction approach described in this thesis, we do this by determining the changes to the website that will be of importance to our approach. In Chapter 7 we describe how these changes are then used to trigger our error-correcting approach which is described in the same Chapter. Chapter 8 contains an example illustrating the application of our error-correction approach. The approach explained in Chapter 7 has been implemented in a proof-of-concept, described in Chapter 9. In the last Chapter we give our conclusions for the work that has been done.

Part I

Background

Chapter 2

Ontologies

The term ontology[1] originates from philosophy where it concerns studies to determine what *entities* and types of entities exist and their *relationships*. In computer science, ontologies were designed to support the *sharing and reuse of formally represented knowledge* among AI systems [11]. Its exactly these properties that have made ontologies popular among other disciplines besides AI research.

We will continue to give a definition for an ontology in the next Section, followed by a discussion of a number of ontology languages and their context.

2.1 Definition

The short answer [11]:

> An ontology is a explicit and formal specification of a shared conceptualization.

Which brings us to the need to specify a *conceptualization*. A conceptualization is an abstract, simplified view of a world, describing it's objects, concepts and other entities and their relationships [12]. Every knowledge based system implicitly or explicitly *commits* to some conceptualization.

Specifying such a conceptualization is building an ontology. It is a description of the *concepts* and *relationships* that can exist for an agent or a community of agents [11]. Since these agents *commit* to using the same, shared, description, they are agreeing to use the same *vocabulary* to exchange queries and assertions among each other.

[1]Wikipedia, Ontology, see `http://en.wikipedia.org/wiki/OntOlogy`

The vocabularies used are constructed using *ontology languages*, these languages share a number of common constructs[2]:

Concepts Anything about which something can be said (e.g., a vertebrate)

Attributes The properties of a concept (e.g., a vertebrate is cracked)

Relationships The relationships among concepts (e.g., a vertebrate is a part of a spine)

In the following Section, the most prominent examples of ontology languages and their context will be described.

2.2 Web Ontology Languages

The ”Web” in ”Web Ontology Languages” means that these languages were designed to be compatible with the WWW in general and the *Semantic Web* in particular[3].
The Semantic Web according to Tim Berners-Lee [18]:

> ”The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.”

The implementation of this Semantic Web happens through layers of Web Technologies and standards built upon each other [19]. Figure 2.1 gives an overview of these layers.
The Unicode and URI [3] layers at the bottom make sure international character sets are used and the objects in the Semantic Web are referable.
The XML layer, combining XML, namespaces and XML-Schema provide Semantic Web compatibility with XML standards.
The layer above that uses RDF and RDF-S (see Section 2.2.1) to respectively make statements about Semantic Web objects and define vocabularies for the Semantic Web.
The Ontology layer adds the ability to add relationships among the Semantic Web objects. This layer uses OWL (see Section 2.2.2) to express these relationships.
The layers above the ontology layer are still the topic of research. The Logic layer

[2]Wikipedia, Computer Science Ontology , see `http://en.wikipedia.org/wiki/Ontology\_(computer\_science)`

[3]McGuinness, D.L, and Frank van Harmelen, F, ”OWL Web Ontology Language Overview”, see `http://www.w3.org/TR/owl-features/`

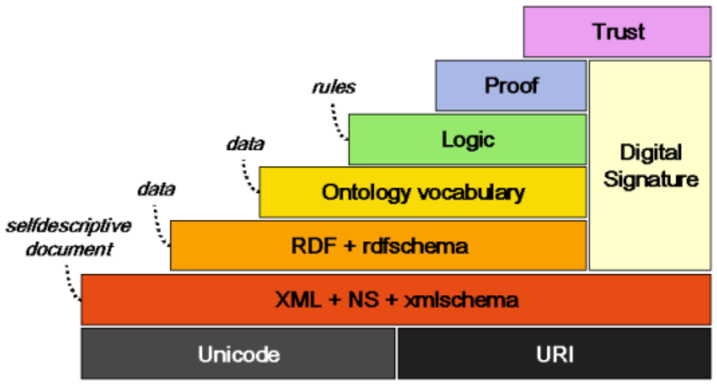


Figure 2.1: The Semantic Web Layers

adds the possibility to express rules, which can be evaluated by the Proof layer and assigned a measure of trust by the Trust layer.

The following Sections will describe the underlying RDF-S framework and **the** web ontology language OWL.

2.2.1 RDF-S

As mentioned in Section 2.2, the *Resource Description Framework* (RDF)[4] allows to represent information about resources on the WWW, using named properties and values. RDF however, provides no mechanisms for describing these properties, nor does it provide a way for describing the relationships between these properties and other resources. That is the role of the RDF vocabulary description language, RDF Schema (RDF-S) . RDF-S defines classes and properties that may be used to describe classes, properties and other resources .

Classes

During any description process, it's necessary to define what kind of things to describe. These "*kinds of things*" correspond to *classes* in RDF-S. A class in RDF-S corresponds to the generic concept of a type or category. Classes can represent almost any kind of thing, such as Web pages, people, document types, databases or abstract concepts. Classes are described using the RDF-S resources `rdfs:Class`, and the properties `rdf:type` and `rdfs:subClassOf`. These properties allow stating that a resource is an instance of a class (even of class "`rdfs:Class`") and stating one class is a subclass of another respectively.

[4]Manola, F., Miller, E., RDF Primer, see `http://www.w3.org/TR/rdf-primer/`

Listing 2.1 gives an RDF/XML example of the classes described by Figure 2.2:

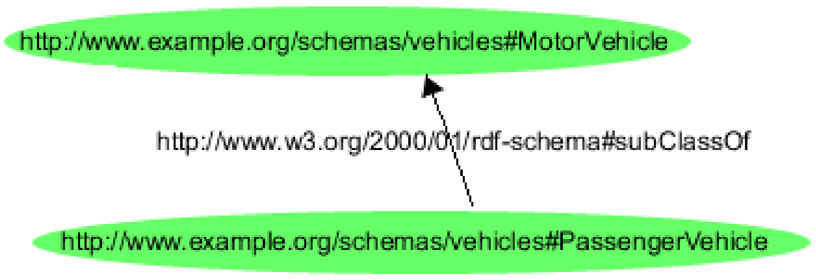

http://www.example.org/schemas/vehicles#MotorVehicle
http://www.w3.org/2000/01/rdf-schema#subClassOf
http://www.example.org/schemas/vehicles#PassengerVehicle

Figure 2.2: The Vehicle Class Hierarchy

```
<rdf:RDF
   xmlns:rdf=”http: //www.w3.org/1999/02/22−rdf−syntax−ns#”
   xmlns:rdfs=”http: //www.w3.org/2000/01/rdf−schema#”
   xml:base=”http: //example.org/schemas/vehicles”>

< rdf:Description  rdf:ID=”MotorVehicle”>
   <rdf:type  rdf:resource =”http: //www.w3.org/2000/01/rdf−
        schema#Class”/>
</ rdf:Description >

< rdf:Description  rdf:ID=”PassengerVehicle”>
   <rdf:type  rdf:resource =”http: //www.w3.org/2000/01/rdf−
        schema#Class”/>
   <rdfs:subClassOf  rdf:resource =”#MotorVehicle”/>
</ rdf:Description >

</rdf:RDF>
```

Listing 2.1: The Vehicle Class Hierarchy in RDF/XML

Properties
In addition to describing the specific classes of things RDF-S allows the description of specific *properties* that characterize those classes of things (such as `registeredTo` to describe a passenger vehicle). In RDF-S, properties are described using the RDF class `rdf:Property`, and the RDF-S properties `rdfs:domain`, `rdfs:range`, and `rdfs:subPropertyOf`.

```
<rdf:Property rdf:ID=" registeredTo ">
  <rdfs:domain rdf:resource ="#MotorVehicle"/>
  <rdfs:range rdf:resource ="#Person"/>
</ rdf:Property >

<rdfs:Class rdf:ID="Person"/>
```

Listing 2.2: The Vehicle Properties in RDF/XML

The RDF-S properties `rdfs:domain` and `rdfs:range` describe the subject and object of the property respectively.

Instances
Now that we have shown how to create descriptions of classes and instances, we can illustrate *instances* of those classes and properties. Listing 2.3 describes an instance of the `ex:PassengerVehicle` class described in Listing 2.1.

```
<rdf:RDF xmlns:rdf="http: // www.w3.org/1999/02/22−rdf−syntax−
    ns#"
            xmlns:ex=" http: // example.org / schemas/ vehicles #"
            xml:base=" http: // example.org / things ">

  <ex:PassengerVehicle rdf:ID="johnSmithsCar">
      < ex:registeredTo  rdf:resource =" http: // www.example.org/
          staffid /85740"/>
  </ ex:PassengerVehicle >
</rdf:RDF>
```

Listing 2.3: A Vehicle Instance in RDF/XML

Note that the `ex:registeredTo` property used in the `ex:PassengerVehicle`, is inherited from the `ex:PassengerVehicle` superclass (see Listing 2.2).

Limitations
RDF-S has a number of limitations to its expressiveness:

- Lack of localized range and domain constraints (e.g.: You cannot express that the range of `hasChild` is person when applied to persons and elephant when applied to elephants)

- Lack of value and cardinality constraints (e.g.: You cannot express that all instances of person have a mother that is also a person, or that persons have exactly 2 parents)
- Lack of transitive, inverse or symmetric properties (e.g.: You cannot express that `isPartOf` is a transitive property, that `hasPart` is the inverse of `isPartOf` or that `touches` is symmetric)
- Lack of constructs for class combination (You cannot express the union, intersection, disjointness or complement of classes)
- Lack of ways to express equivalence between classes, properties and instances
- Lack of native support for *reasoning*

To remedy these limitations, OWL, which is described is the next Section, was developed on top of RDF-S.

2.2.2 OWL

OWL, the *Web* Ontology Language was designed to enable machine interpretation of *Web* contents. OWL is an extension of RDF-S, adding more vocabulary for describing properties and classes: among others, relations between classes, cardinality, equality, richer typing of properties, characteristics of properties, and enumerated classes. As shown in Table 2.1, OWL provides three increasingly expressive sub languages designed for use by specific communities of implementers and users[5].

OWL Full
OWL DL (Description Logics)
OWL Lite

Table 2.1: An RDF Statement

OWL Lite supports simple classification hierarchy and simple constraints (e.g. cardinality constraint values are limited to 0 and 1). It is the easiest to implement and even provides a quick migration path for thesauri and other taxonomies.

[5]McGuinness, D.L, and Frank van Harmelen, F, "OWL Web Ontology Language Overview", see `http://www.w3.org/TR/owl-features/`

OWL DL offers maximum expressiveness while retaining computational completeness and decidability. OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (e.g., while a class may be a subclass of many classes, a class cannot be an instance of another class). Full formalism of *Description Logics* is supported.

OWL Full offers maximum expressiveness with full syntactic liberty of RDF, but without computational guarantees (e.g., a class can be treated simultaneously as a collection of individuals and as an individual in its own right).

The following Sections provide an overview of the most important OWL constructs.

Classes

Listing 2.4 gives an example of an OWL class definition for a Wine ontology. Every individual in the OWL world is a member of the class `owl:Thing`. Thus each user-defined class is implicitly a subclass of `owl:Thing`. Domain specific root classes are defined by simply declaring a named class[6].

```
<owl:Class rdf:ID=”Wine”>
  <rdfs:subClassOf  rdf:resource =”&food:PotableLiquid”/>
</owl:Class>

<owl:Class rdf:ID=”Pasta”>
  <rdfs:subClassOf  rdf:resource =”#EdibleThing” />
</owl:Class>
```

Listing 2.4: An OWL Wine Class.

This example (Listing 2.4) defines two classes, `Wine`, a subclass of `PotableLiquid` and `Pasta`, a subclass of `EdibleThing`.

Individuals

Individuals are used to describe the members of classes. OWL individuals are introduced by declaring them to be a member of a class. Listing 2.5 shows an OWL class definition and an individual of that class[7].

[6]Smith, M.,K., Welty, C., McGuinness, D.L. 2004, ” OWL Web Ontology Language Guide”, see `http://www.w3.org/TR/owl-guide/`

[7]Smith, M.,K., Welty, C., McGuinness, D.L. 2004, ” OWL Web Ontology Language Guide”, see `http://www.w3.org/TR/owl-guide/`

```
<owl:Class rdf:ID=”WineGrape”>
  <rdfs:subClassOf  rdf:resource =”&food:Grape” />
</owl:Class>

<WineGrape rdf:ID=”CabernetSauvignonGrape” />
```

Listing 2.5: An OWL WineGrape Class and Instance.

The individual `CarbernetSauvignonGrape` is a valid individual because it denotes a single `WinGrape` variety.

Properties
OWL properties are binary relations that allow the assertion of general facts about class members and specific facts about individuals. Two types of properties are distinguished:

Datatype Properties describe relations between instances of classes and RDF literals and XML Schema datatypes.

Object Properties describe relations between instances of two classes.

Listing 2.6 extends the Wine ontology with a property for the `Wine` class.

```
<owl:ObjectProperty  rdf:ID=”madeFromGrape”>
  <rdfs:domain  rdf:resource =”#Wine”/>
  <rdfs:range   rdf:resource =”#WineGrape”/>
</owl:ObjectProperty>
```

Listing 2.6: An OWL madeFromGrape property.

The `madeFromGrape` property shown in Listing 2.6 relates instances of the class `Wine` to instances of the class `WineGrape`. Properties allow the restriction of their classes, as show in Listing 2.7.

```
<owl:Class rdf:ID=”Wine”>
  <rdfs:subClassOf  rdf:resource =”&food:PotableLiquid”/>
  <rdfs:subClassOf>
    < owl:Restriction >
      <owl:onProperty  rdf:resource =”#madeFromGrape”/>
```

```
        <owl:minCardinality   rdf:datatype =”&xsd:nonNegativeInteger
              ”>1</owl:minCardinality>
      </ owl:Restriction >
   </rdfs:subClassOf>
    ...
</owl:Class>
```

Listing 2.7: `Wine` is made from at least one `WinGrape`.

The cardinality constraint enforced in Listing 2.7 makes sure that ”Wine is made from at least one Grape”. The class Restriction used in the above example defines an *anonymous* class that represents the set of things with at least one `madeFromGrape` property. Because `Wine` is a subclass of this set, the restriction enforces a cardinality constraint on the `madeFromGrape` property, meaning that each individual `Wine` must participate in at least one `madeFromGrape` relation.

Of course, many more characterizations are possible, including:

- cardinality (e.g. `minCardinality`, `maxCardinality`, `cardinality`)
- equality (e.g. `equivalentClass`, `equivalentProperty`, `sameAs`)
- relationships between classes (e.g. `disjointWith`, `unionOf`)
- characteristics of properties (e.g. `FunctionalProperty`)
- etc...

A full definition of the OWL syntax can be found in the OWL Web Ontology Language Guide[8].

[8]Smith, M.,K., Welty, C., McGuinness, D.L. 2004, ” OWL Web Ontology Language Guide”, see `http://www.w3.org/TR/owl-guide/`

Chapter 3

Ontology Based Website Design Methods

Modern websites are more closely related to large-scale applications than to the network of linked pages they were in the early years of the WWW. Back then, websites were simple enough to be designed in an ad-hoc manner. Applying this ad-hoc manner to design complex and large web-applications is likely to result in a number of usability and maintainability problems (see Section 1.1) [10].

To avoid these problems, a large number of structured Website Design Methods have been developed (e.g., WebML[1], OOHDM [20], OOH [21], UWE [22], RMM [23]). Due to their structured nature, these methods make use of *models* to capture the complexity of the design process. In the past, model representation varied from graphical representation techniques (e.g. ER [24], UML [25], ORM [36]), over plain text to more formatted descriptions (XML[2]). With the rise of the Semantic web, ontologies (RDF-S[3], OWL[4], see Section 2) have become an increasingly popular way to provide formal descriptions of complex information. Some Website Design Methods have evolved to make use of ontologies and the opportunities they provide (e.g. Hera [16], WSDM [26], SHDM [32], OntoWeaver [17], OntoWebber [27] and XWMF [28]) .

In the next Sections, the four most current ontology based website design methods are reviewed, namely WSDM, Hera, OntoWeaver and finally SHDM.

[1]The Web Modeling Language, see `http://webml.org`

[2]Extensible Markup Language (XML), see `http://www.w3.org/XML/`

[3]Manola, F., Miller, E., RDF Primer, see`http://www.w3.org/TR/rdf-primer/`

[4]Web Ontology Language (OWL), see `http://www.w3.org/2004/OWL/`

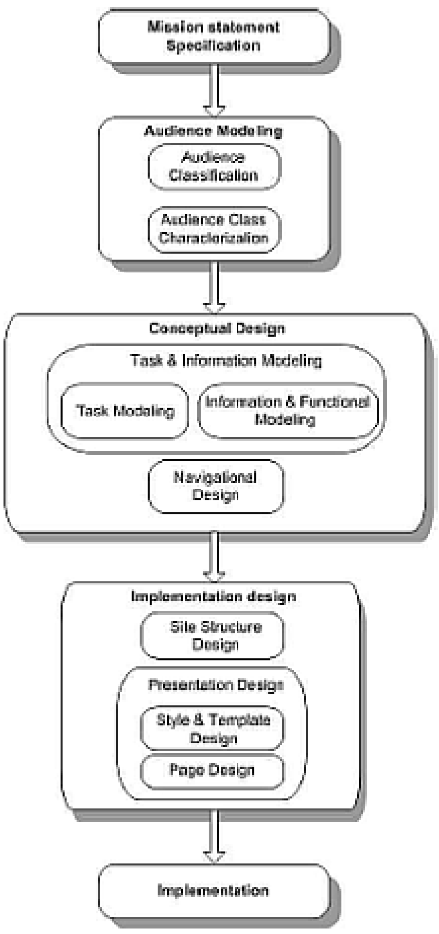


Figure 3.1: Overview of the WSDM phases.

3.1 WSDM

The Website Design Method (WSDM) project is led by Prof.Dr. Olga De Troyer, head of the WISE research group[5] at the Vrije Universiteit Brussel[6]. WSDM is an audience-driven design method, meaning that the requirements of the intended users form the starting point for the method, in contrast with most design methods, which are data-driven. Figure 3.1 (from [34]) gives an overview of the different phases of the WSDM design process. The models of each of these phases have

[5]Web & Information Systems Engineering, see `http://wise.vub.ac.be/`
[6]Vrije Universiteit Brussel, see `http://www.vub.ac.be`

been provided with explicit formalizations, using ontologies [15].
The WSDM method is iterative in nature, not completely sequential as suggested by the Figure.

Each of the phases of Figure 3.1, will be described in the following Sections, followed by a description of the adaptive capabilities of WSDM.

3.1.1 Mission Statement Specification

The intention of this phase is to identify the purpose of the website, as well as the subject and the target users. This phase creates the mission statement, which is formulated in natural language.

3.1.2 Audience Modeling

The audience modeling phase is divided in two sub-phases: Audience Classification followed by Audience Characterization.

3.1.2.1 Audience Classification

During this phase, the target users identified in the mission statement are refined into audience classes. Users with the same information and functional requirements become members of the same audience class. Users with additional requirements form audience subclasses. This results in a hierarchy of audience classes [15].

3.1.2.2 Audience Characterization

During audience characterization, relevant characteristics are collected for each audience. Examples of characteristics are age, experience level and language. These characteristics are taken into account when deciding the structure and presentation for the *navigation track* of each audience class.

The model representing the audience class hierarchy together with for each audience class their characteristics and their set of requirements is called the audience model.

3.1.3 Conceptual Design

Conceptual Design has two sub-phases, the Task & Information Modeling and the Navigational Design phases. The content and functionality at a conceptual level are defined during the Task Modeling phase and the conceptual navigational structure is defined during the Navigational Design. The resulting conceptual design does not contain any implementation details or commitment to a target platform.

3.1.3.1 Task & Information Modeling

The Task & Information Modeling phase is in turn composed of two sub phases, namely the Task Modeling sub phase and the Information & Functional Modeling sub phase.

3.1.3.1.1 Task Modeling

Each requirement of each audience class from the Conceptual Model results in a task that is part of the Task Model. These tasks will describe the information and processes needed to fulfill the requirements causing them. The tasks are decomposed into elementary tasks. Figure 3.2 (from [34]) gives an example of a task decomposition. The task being decomposed here is the ”`Collect Items`” task for an online bookstore application. The model itself is expressed using an adapted version of the Concurrent Task Trees (CTT) [35] notation, called CTT+. These elementary tasks are then used in the next sub phase.

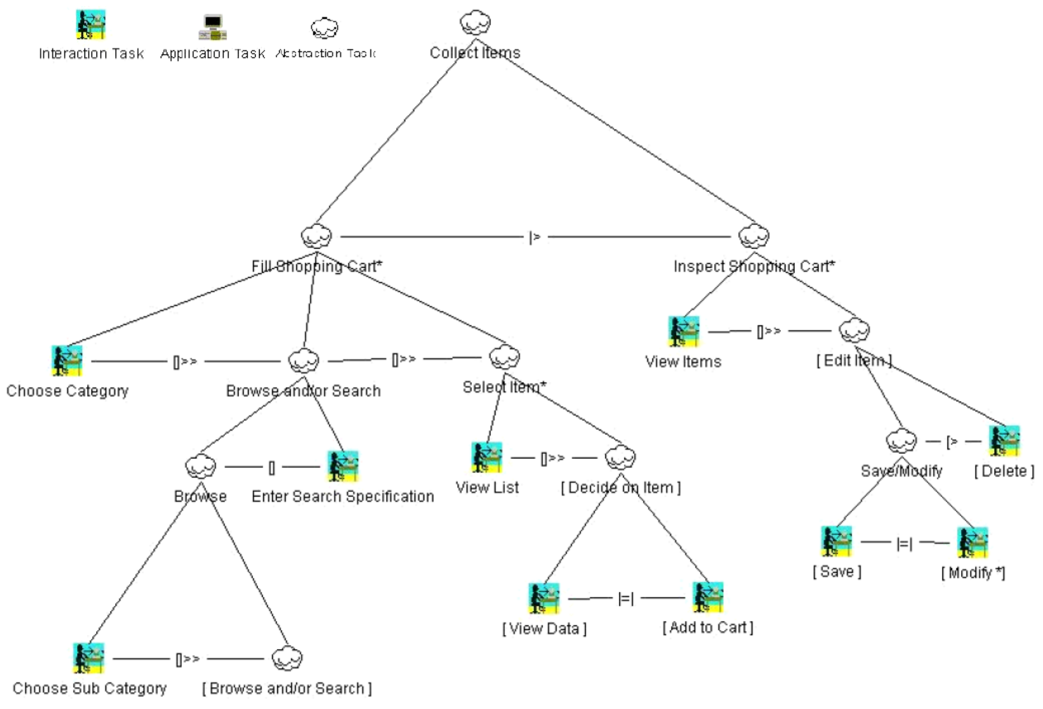


Figure 3.2: A WSDM CTT+ Task Model.

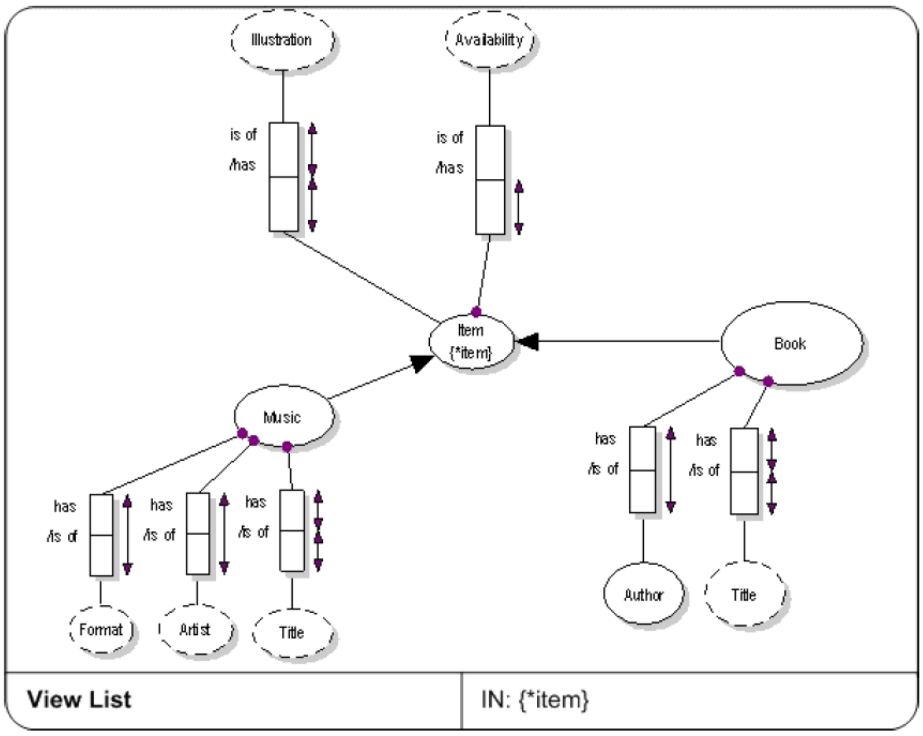


Figure 3.3: A WSDM Object Chunk.

3.1.3.1.2 Information & Functional Modeling

The Information & Functional Modeling phase creates for each of the tasks in the task model an *object chunk*. Such an object chunk is a data model describing the necessary information and functionality needed to fulfill the requirement causing the elementary task.
Object Role Modeling (ORM) [36] is used to *visually* represent the object chunks. Figure 3.3 gives an example of an object chunk for the `View List` task from the example in Figure 3.2, represented in ORM.
OWL is used by WSDM to *internally* represent these object chunk models and add a conceptual annotation to the Task Modeling phase. This results in the creation of a number of links between object chunks and the concepts of one or more ontologies. The conceptual annotation adds the semantic meaning of these object types and roles to the design method [14].

3.1.3.2 Navigational Design

The goal of the Navigational Design is to define how the members of the different audience classes can navigate through the website and perform their tasks. For each audience class, an navigational audience track is created. An audience track can be considered as a sub site with all and only the information and functionality needed by the associated audience class members.

All audience tracks are combined into the *Conceptual Structure* by means of *structural links*. The same structural links are used to group the *task navigational models* into the audience tracks. Task navigational models contain these elements [34]:

Components These are placeholders for object chunks, allowing the usage of the same object chunk in different contexts (tasks).

Links These are links between components that are used to express the workflow or process logic, as expressed in the task model by means of the CTT relations.

Figures 3.4 and 3.5 (from [34]) provide examples of subtask Navigational Models. These are subtasks from Figure 3.2.

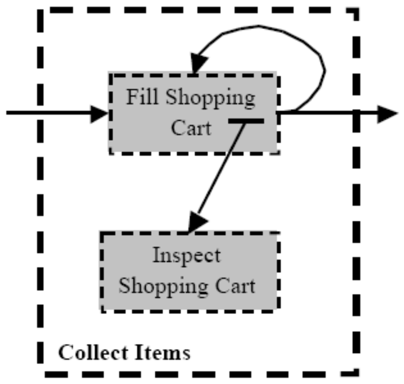


Figure 3.4: Subtask Navigation Model for `Collect Items`

3.1.4 Implementation Design

During the implementation design phase, the conceptual design models are extended with information required for the actual implementation. This design phase is divided in two sub phases:

3.1.4.1 Site Structure Design

The Site Structure Design phase decides which component(s) and links defined in the navigational model will be grouped onto web pages. Different site struc-

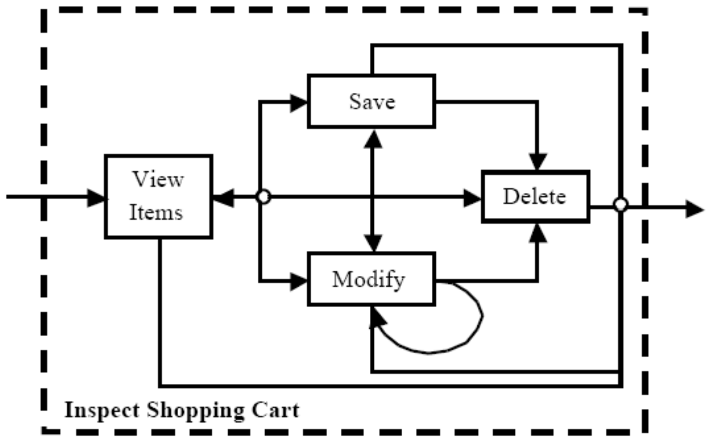


Figure 3.5: Subtask Navigation Model for `Inspect Shopping Cart`

tures can be defined for the same conceptual structure, to target different devices, contexts or platforms. The result of this phase is the site structure model.

3.1.4.2 Presentation Design

The Presentation Design phase defines the look and feel of the website, as well as the positioning of page elements. This phase has two sub phases:

Style & Template Design This sub phase handles the design of different page templates for the website, as well as the style for the page-elements.

Page Design This sub phase describes how the components assigned to a page are presented. The layout of each page is based on one of the templates defined during the previous sub phase. The editable regions specified in these templates need to be filled in using presentation concepts (see Section 5.4 on Page 69).

The output of these phases is the presentation model which consists of a set of templates and a page model for each page of the site structure model [15].

3.1.5 Implementation

WSDM enables the automatic generation of the actual implementation from the information collected during the different design phases. A Transformation Pipeline has been defined, which takes as input the object chunks and WSDM models and outputs the implementation for the chosen platform and implementation language. Figure 3.6 gives an overview of the WSDM architecture and the combination

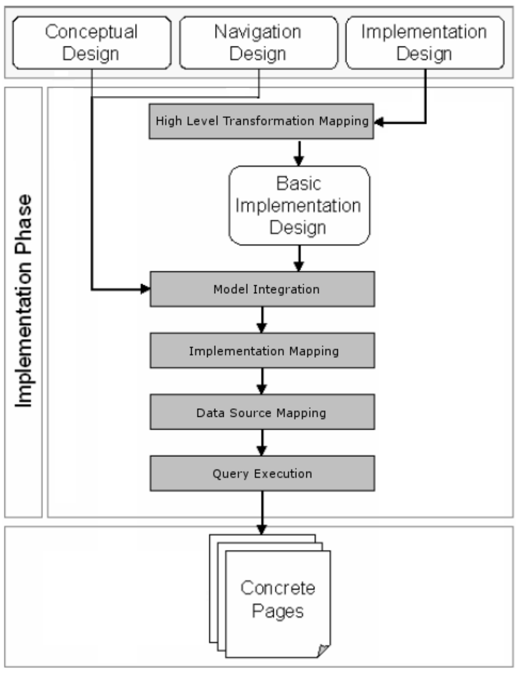


Figure 3.6: An Overview of the WSDM Architecture

of the different models to reach the automatic implementation generation. The phases from the architecture are:

High Level Transformation Mapping The high level presentation concepts (see Section 5.4) and templates are translated into primitive presentation concepts.

Model Integration The navigation, page structure, template & style and page design models are integrated into one single model.

Implementation Mapping The integrated model is partially transformed toward the chosen platform (e.g. HTML).

Data Source Mapping The references to attributes in the object chunks are resolved and mapped to their data source by executable queries.

Query Execution The queries are executed and the resulting data can be inserted into the actual pages.

3.1.6 Adaptation

Adaptation in WSDM happens through a ”Web site Overlay Model” [41]. This Overlay Model supports the storage of website usage information at runtime. Additionally, a high level rule based adaptation specification language (ASL) for WSDM has been developed to allow a number of operations:

information storage operations These operations allow the population of the Web site Overlay Model (e.g. adding tracking variables to design elements, updating the value of tracking variables).

model transformation operations These operations allow the manipulation of the WSDM navigational and page design models and thus, the adaptation of the web site design;

ASL allows the web designer to specify about which (design) element information needs to be stored, when and how to update this information and when and how to perform the model transformation operations. These specifications use the following concepts;

Adaptation Strategies and Policies Adaptation Strategies specify rules defining which adaptive behavior needs to be performed; Adaptation Policies specify the events when the Adaptation Strategies need to be performed.

Events User generated events (e.g. clicking a link, visiting a node,) or System generated events (elapse of a time interval, tracking variables attaining a certain value,) trigger adaptation strategies.

Rules Rules can be used to specify iteration, conditional execution of an action, add tracking variables, declare variables, assign values to (tracking) variables or perform pre-defined operations.

Control flow Control flow operations add programming logic to adaptation strategies (e.g. loop-constructs, if-then).

Expressions and set-expressions Expressions allow the designer to express calculations with tracking variable values and arbitrary values. Arithmetic, statistical and set theory operators are provided.

Using these concepts, ASL allows the designer to specify, at design time, which adaptation (on the design models) can be performed at runtime.

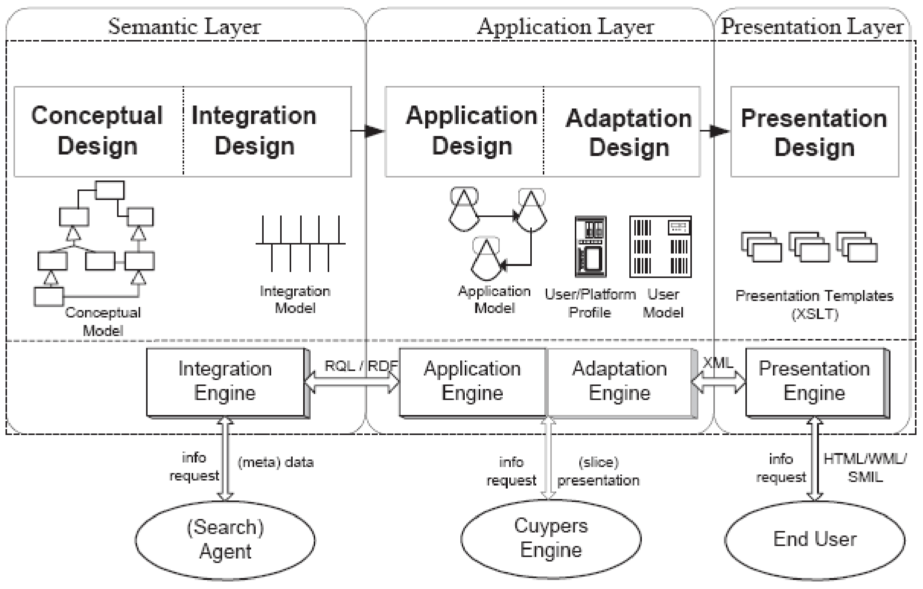


Figure 3.7: The WIS architecture in Hera.

3.2 Hera

The Hera Research Program[7], located at the Vrije Universiteit Brussel[8] and the Technische Universiteit Eindhoven[9], has developed the design method of the same name. Hera is a model-driven design method focusing on the development of context-dependent or personalized *Web Information Systems* (WIS).
Based on the principle of *separation of concerns* it distinguishes three design steps: conceptual design, navigational design and presentation design. At each design step different aspects of adaptation and personalization can be specified in terms of *formal models* for *data transformations*.
Figure 3.7, taken from [16] gives an overview of the three layers that make up the architecture of WIS in Hera.
These three layers are:

The Semantic Layer This layer defines the content that is managed in the WIS in terms of a conceptual model. This layer includes the definition of the integration process needed to gather the data from different sources. The data can originate from databases, or, if the data is made available from outside the WIS, a search agent or information retrieval engine could be the interface to the WIS.

[7]The Hera research program, see `http://wise.vub.ac.be/hera`
[8]Vrije Universiteit Brussel, see `http://www.vub.ac.be`
[9]Technische Universiteit Eindhoven, see `http://www.tue.nl`

The Application Layer This layer defines the navigation view on the data in terms of an application model, which represents the structure shown to the user in the hypermedia presentation. This layer includes the definition of the adaptation in the hypermedia generation, meaning it provides a view on the conceptual model to satisfy the needs of a specific user class.

The Presentational Layer This layer defines the presentation details that together with the definitions from the Application Layer are needed for the generation of a presentation to the user's browsing platform.

The top part of Figure 3.7 describes the Hera design methodology, while the bottom part describes the Hera suite, which makes use of the models resulting from the design method to create the WIS.

Each of the three layers of the WIS architecture corresponds with a step of the Hera design method and the models it creates. The conceptual design results in the conceptual model provided to the semantic layer, the navigational design creates the application model for the application layer and the presentational design provides the presentation model to the presentation layer.
RDF-S (see Section 2.2.1) is used to represent both the conceptual and application model, while the presentation model is created as a XSLT transformation from the conceptual model.

The following Sections will describe the three design steps of the Hera design method.

3.2.1 Conceptual Design

3.2.1.1 Conceptual Model

The conceptual design step creates the conceptual model (CM). Figure 3.8 from [16] shows an example of a conceptual model for a virtual art gallery.

The CM is composed of concepts and concept properties that together define the domain ontology. There are two types of concept properties: concept attributes which associate media items to the concepts and concept relationships that define associations between concepts. As shown in Figure 3.8 the RDF-S used to express the CM is extended with two descriptions [16]:

CM Properties These describe the `inverse` and `cardinality` of concept relations:

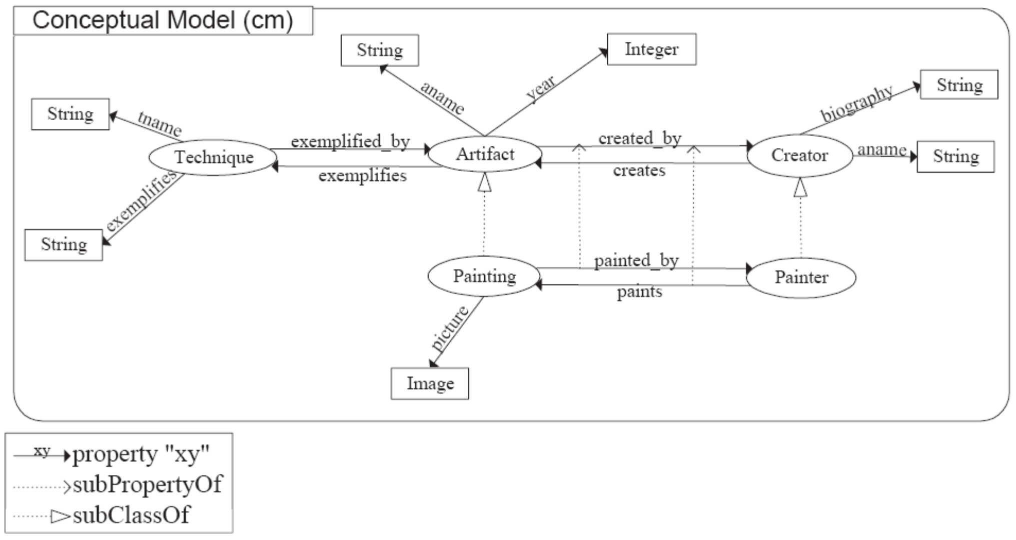


Figure 3.8: An RDF-S Conceptual Model for a virtual art gallery.

System Media Types These describe the possible concept types using the multimedia ontology. This extensible ontology contains types as `text` and `image`.

Adaptation in the CM is based on the conditional inclusion of elements. XSLT conditions referencing (ranges of) data, describing the device capabilities and user preferences can be used to achieve this conditional inclusion. These conditions are then stored in a user profile [30].

3.2.1.2 Navigation Data Model

The CM can be extended with a Navigation Data Model (NDM) [29]. The purpose of this model is to complement the CM with a number of auxiliary concepts which can be used in the AM when defining the *behavior* of the application and its navigation structure. Figure 3.9 (taken from [29]) shows the NDM for a virtual art gallery's shopping trolley. The NDM captures these concepts:

SelectedPainting This represents the Paintings the user *selected* from the form.

Order This represents a single ordered item, such an item `includes` a `SelectedPainting` and has a `quantity`

Trolley This represents a shopping trolley which `contains` several `Orders`.

From the system perspective the concepts in the NDM can be divided into two groups:

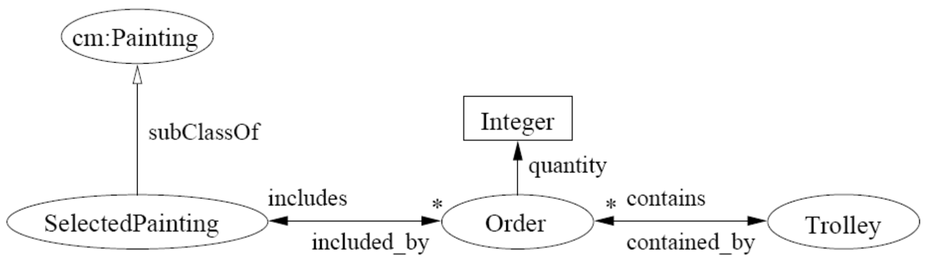


Figure 3.9: A Navigation Data Model for a virtual art gallery.

- Concepts representing views over the concepts from the CM (e.g., `SelectedPainting` from the NDM is a view of a `Painting` from the CM). Instantiation can only happen for subsets of instances of CM concepts.
- Concepts forming a local repository. Instantiation happens based on the users interaction, meaning that the data is created, updated, and potentially deleted on-the-fly.

The instantiation of both groups of concepts is triggered by a certain action (such as pressing the submit button) specified in the Application Model.

3.2.2 Application Design

The application design step creates an application model (AM). The AM describes the navigational aspects of the hypermedia presentation [16].

3.2.2.1 Application Model

Figure 3.10 gives an example of an AM for the virtual art gallery. An AM is composed of slices and slice properties that define the navigation ontology. A slice is a meaningful presentation unit of some *media items*. These media items may originate from different CM concepts. A slice is always owned by a concept from the conceptual model and each slice has attributes that correspond to attributes in the conceptual model. There are two types of slice properties:

slice composition a slice encloses another slice.

slice navigation a slice points to another slice by means of a Hyperlink.

Primitive slices contain only one media item. Higher level slices use slice composition to contain other slices. Top-level slices correspond to pages presented to the user [16].

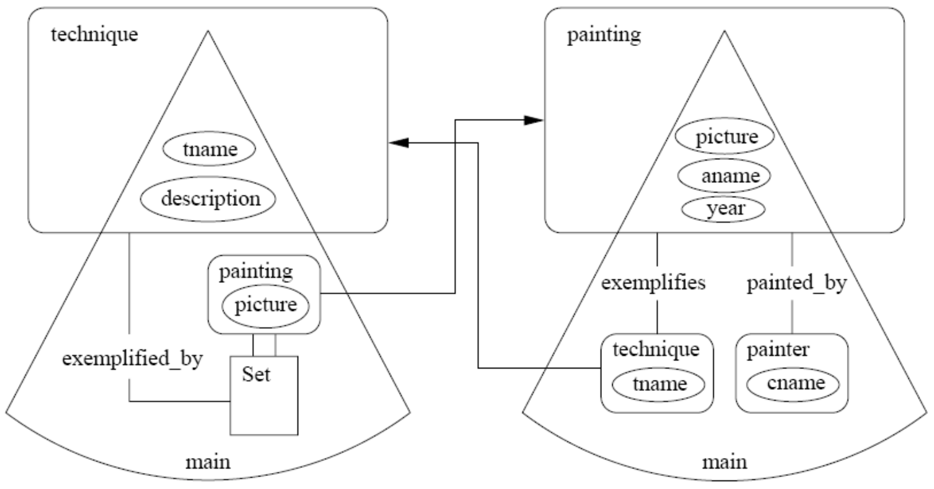


Figure 3.10: An RDF-S Application Model for a virtual art gallery.

The example from Figure 3.10 shows an AM that is composed of two slices (`technique` and `painting`, subclasses of `Slice`) and two slice navigation properties between these two slices. The primitive slices are shown as ovals and the slice composition properties are shown through nesting. Because a single `technique` is used for many `paintings`, a set of links is created when navigating from technique to painting. Practically, the two top-level slices `technique` and `painting` correspond to two webpages.

Adaptation for the AM is reached by associating appearance conditions to slice references. Figure 3.11 shows the previous AM, with extra appearance conditions added.
These appearance conditions allow for static and dynamic adaptation, done prior and during the presentation browsing respectively.

3.2.2.2 Extended Application Model

Like the NDM provides extended information for the CM, the AM has an Extended Application Model (EAM) [29]. The EAM allows to add:

- State information (e.g., information received from a user query).
- Input controls with navigation information (e.g., selecting paintings for a trolley).

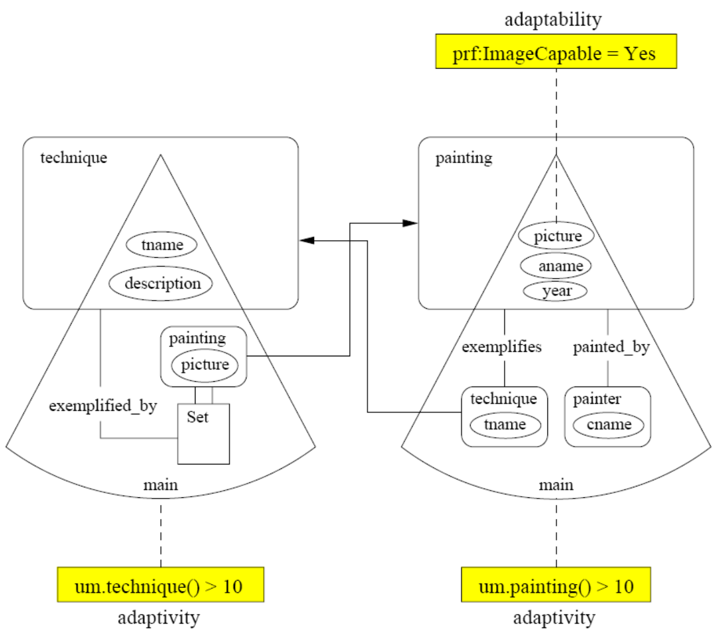


Figure 3.11: An Adaptive Application Model.

The forms containing the input controls are implemented using XFORMS[10] and the queries use the Sesame RDF Query Language (SeRQL)[11].

Figure 3.12(taken from [29]) gives an example of and EAM for a virtual art gallery, containing forms an queries.
The forms are added to slices (e.g., `BuyForm` and `SelectForm`) and have a number of properties associated with them:

The Owning Concept Nesting is used to indicate the concept owning the form (e.g., a `SelectForm` is selecting `Paintings`):

The Input Controls These controls allow the user to input information in the form (e.g., selecting a painting in `SelectForm` and indicating the quantity in `BuyForm`)

The Outgoing Navigational Relationships Forms result in navigation to other slices (e.g. both `BuyForm` and `SelectForm` result in navigation (`Q1` and `Q2`) to the slice containing `Trolley`)

[10]XForms - The Next Generation of Web Forms, see `http://www.w3.org/MarkUp/Forms/`
[11]Sesame RDF Query Language, see `http://openrdf.org/`

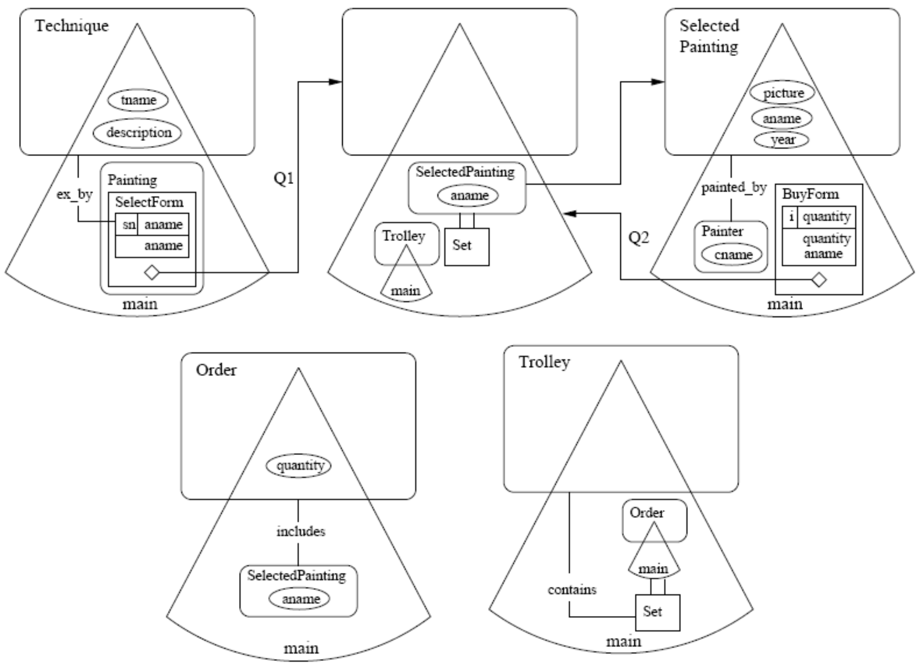


Figure 3.12: An Extended Adaptive Application Model.

These Navigational Relationships are in fact queries, passing state from one slice to another.

3.2.3 Presentational Design

The Presentational Design step produces the Presentation model (PM) and this model describes how the data is presented to the user. The PM uses the AMACONT project to generate presentations for the user. AMACONT is a component-based document format for building Web applications by linking configurable document components. There are three types of components [30]:

Media Components These components encapsulate concrete media assets by describing them with technical metadata.

Content units These components group media components by declaring their layout in a device-independent way.

Document Components These components define a hierarchy out of content units to fulfill a specific semantic role.

Figure 3.13(taken from [30]) gives an example of a PM for the `technique` slice. The Figure contains conditions and a number of *`BoxLayouts`*. A `BoxLayout`

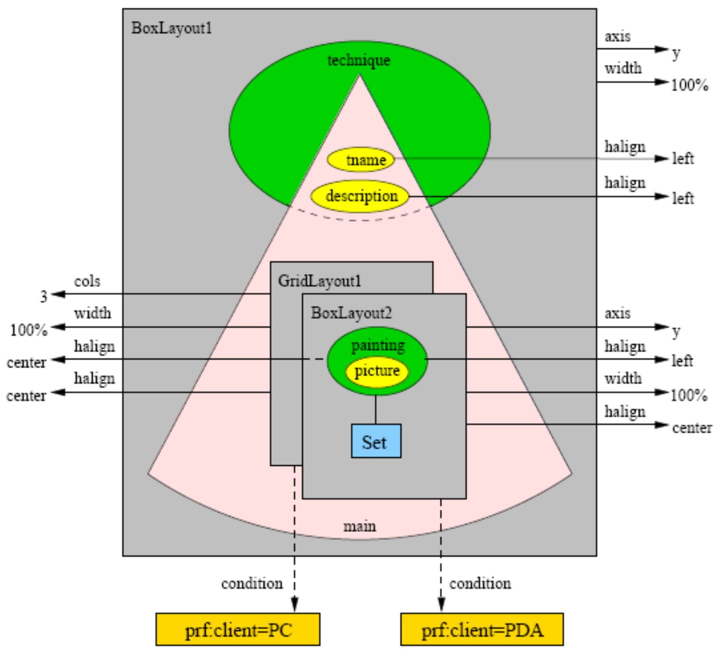


Figure 3.13: A Hera Presentation Model.

is an AMACONT *layout manager*. AMACONT allows the association of XML-based layout descriptions to components, such descriptions are called layout managers. Currently four layout managers are defined:

BoxLayout Lays out multiple components either vertically or horizontally.

BorderLayout Arranges components to fit in five regions: north, south, east, west, and center, resembling the structure of many Web pages consisting of a header, an optional footer, a main area and one or two sidebars.

OverlayLayout Allows to present components on top of each other.

GridLayout Enables to lay out components in a grid with a configurable number of columns and rows.

In the example of Figure 3.13, the layout manager has been assigned to a slice of the AM. The layout of the attributes and subslices of the slice can in turn be positioned using the subcomponent attributes of the component manager [30].

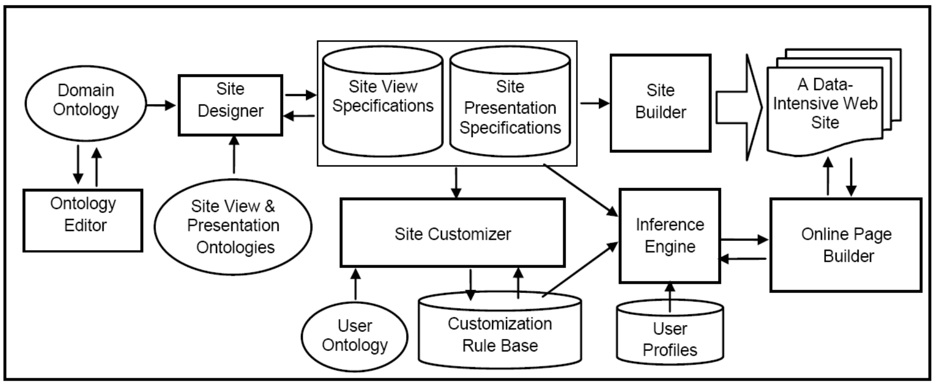


Figure 3.14: The OntoWeaver framework.

3.3 OntoWeaver

The OntoWeaver methodology [37] has been developed at the Knowledge Media Institute (KMI)[12] of the United Kingdom's Open University (OU)[13]. Ontoweaver has been built on top of the Intelligent Information Presentation System (IIPS) [39], also developed by the KMI.

The OntoWeaver method is an ontology-based approach to the design and management of *customized* data-intensive web sites. The OntoWeaver *framework* starts from a domain ontology and aims to produce a customized, personalized data-intensive website. Figure 3.14 (from [17]) gives an overview of this framework.

The Ontoweaver *method* creates the models (ontologies) involved in the framework of Figure 3.14, this method consists of the following steps [37]:

1. Designing the domain ontology;

2. Specifying navigation structures and composing the user interfaces;

3. Defining layouts and presentation styles;

4. Expressing customization requirements.

Each of these steps will be described in the following Sections.

[12]Knowledge Media Institute, see `http://kmi.open.ac.uk/`
[13]Open University distance learning, see `http://www.open.ac.uk/`

3.3.1 Domain Ontology Design

The result of this step is the Domain Ontology. The domain ontology abstracts the underlying domain data model, by means of a set of classes and properties. Figure 3.15 (taken from [17]) gives a graphical example of such an ontology for the KMI web portal.

Part (a) of Figure 3.15 shows the hierarchical structure of the domain ontology

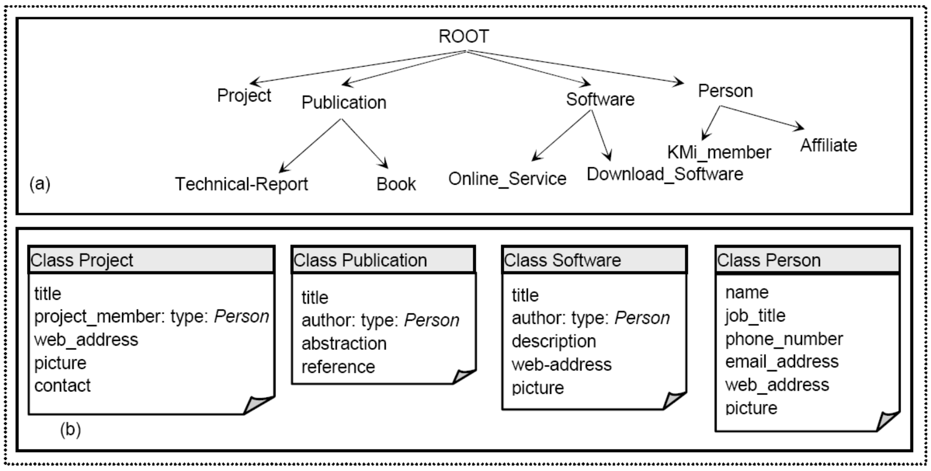


Figure 3.15: Two views of an OntoWeaver domain ontology.

and part (b) shows class descriptions.

The graphical representation of Figure 3.15, corresponds to the actual Domain Ontology, which is expressed in RDF(S). The instantiation of this domain ontology forms the information about a specific KMI web portal version.

3.3.2 Navigational Structure Specification and User Interface Composition

This step is divided in two sub-steps [17]:

Modeling Site Structures This step identifies web pages and defines their purpose and link relationships:

Composing User Interfaces These User Interfaces are composed for each web page from the Structure Model.

These sub-steps are described in the following sub-sections:

3.3.2.1 Modeling Site Structures

The Site Structure Model, that is the result of this design step, is a coarse-grained level structure of an entire web site, which comprises a root page, a number of page nodes and the URI of the underlying domain ontology. Figure 3.16 (from [17]) gives an example sitestructure for the KMI portal.
The site structure in Figure 3.16 consists of page nodes and two kinds of links:

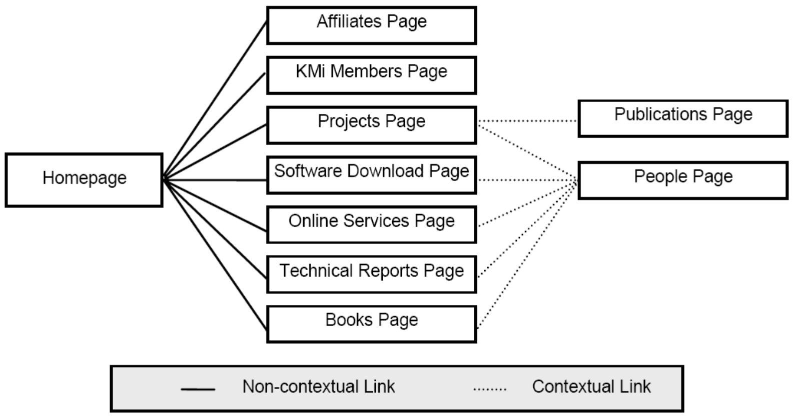


Figure 3.16: An example site structure.

Non-Contextual Links These links simply provide navigation.

Contextual Links These links carry contextual information when they are used to navigate from one webpage to another.

3.3.2.2 Composing User Interfaces

In the OntoWeaver method, web pages are composed of a set of components, and each component is in turn composed of a set of atomic user interface elements and subcomponents. This allows the expression of complex user interfaces. Each of the user interface elements has an URI for identification, which can be referenced by other composite user interface elements to allow re-use [17].
Dynamic features like information publication, querying and user input are supported by the Ontoweaver method.
Note that in this step of the design process, the organization and look and feel of the user interface elements is not yet considered.

The specified navigational structures and composed user interfaces from the previous steps are expressed using the Site View Ontology. Figure 3.17 (from [17])

gives an overview of the site view ontology.
This ontology models a web site as a collection of resources, these logical re-

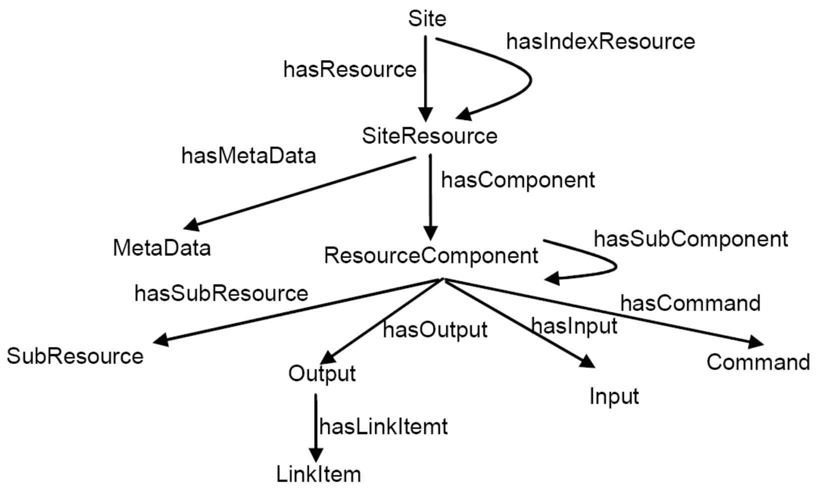


Figure 3.17: An overview of the OntoWeaver Site Ontology.

sources describe web pages, components and user interface elements. The ontology also contains a set of navigational constructs to allow the specification of complex navigational structures.

3.3.3 Layout and Presentation Style Definition

At this design step, the visual appearances and organizations for user interface elements of web pages are specified in terms of the presentation ontology [38]. This ontology provides the following:

- A set of templates to abstract the visual appearance of user interface elements,
- A set of layouts to model the organization features of user interface elements
- A construct called `Presentation` to attach templates to user interface elements
- A construct called `SitePresentation` to group presentation styles and layouts together in a presentation model.

Re-usable templates are used to specify the visual appearance of user interface elements.
Figure 3.18 (from [37]) gives an overview of the presentation ontology.

Two different layout constructs are present in the presentation ontology to model

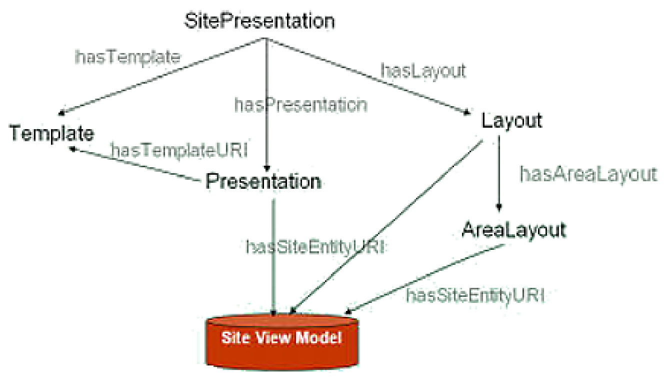


Figure 3.18: An overview of the Presentation Ontology.

the typical layout for user interface elements:

TextLayout This models the layout of atomic user interface elements in terms of alignment within a component:

ComponentLayout This organizes the sub-elements of the component into five sub-areas, top, left, middle, right and bottom. Each of these sub-areas can display user interface elements in a horizontal or vertical layout direction.

3.3.4 Customization Requirements Specification

OntoWeaver contains a framework for customization, which is based on four components [38]:

1. The *User Ontology*, which provides means to describe the user model in the domain specific context;

2. The *Customization Rule Model*, which enables the specification of customization rules;

3. The Site View Ontology and the Presentation Ontology, which provide the declarative site models;

4. The *inference engine*, which reasons about site specifications with customization rules according to the facts of the user profiles.

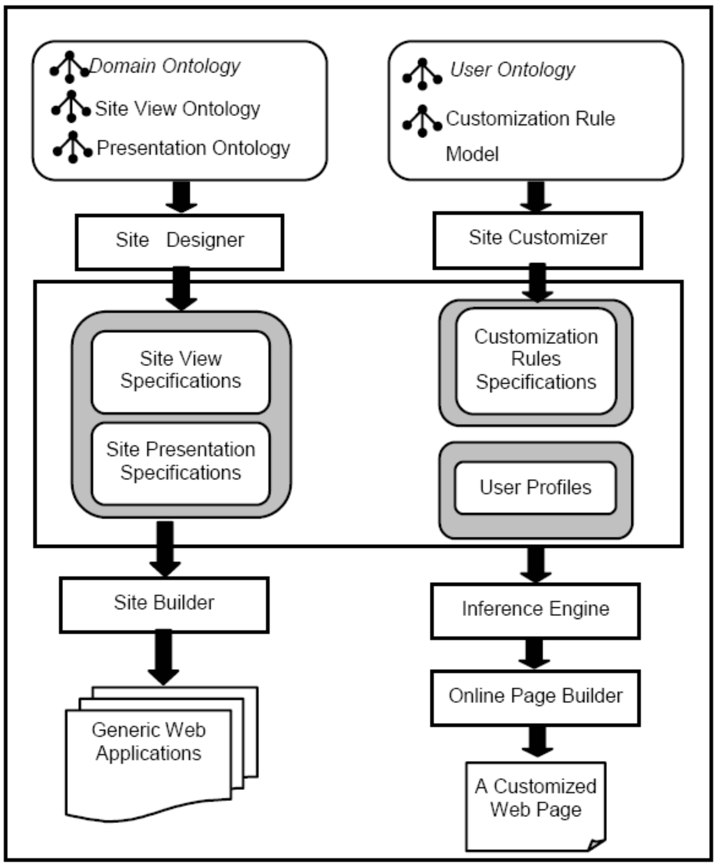


Figure 3.19: An overview of the Customization Framework.

Figure 3.19 provides an overview of the customization framework.

The following Sections will describe the User Model and the Customization Rule Model which are used by the inference engine to create a customized web page.

3.3.4.1 User Model

The user model, that is the result of this sub-phase, describes information about end users. The user model enables [38]:

- The definition of customization conditions at a high level of abstraction;
- Through its instantiations (called User Profiles), the evaluation of the conditions of customization rules and the decision whether the rules should fire.

3.3.4.2 Customization Rule Model

OntoWeaver has dynamic customization support for the target web applications, through the usage of a number of customization rules. These customization rules specify *conditions* determining when certain customizations should happen, and *actions* that realize the customizations.

The OntoWeaver rule model has a condition part and an action part. The condition part describes a condition that has to be satisfied for the customization to take place. The action part describes the adaptation actions, meaning the adding, hiding or modifying of components, or setting presentation or layout properties for components [38].

3.4 SHDM

The Semantic Hypermedia Design Method (SHDM) [32] is based on the Object Oriented HyperMedia Design Method (OOHDM) [20] and both are developed at the Universidade Católica de Brasíia[14]. SHDM extends the OOHDM approach, by using ontology languages to represent the OOHDM models. The SHDM Method is a model-driven design method which uses five steps [32]:

1. Requirements Gathering
2. Conceptual Design
3. Navigational Design
4. Abstract Interface Design
5. Implementation

Each step focuses on a particular aspect and produces models, describing details about an application to be run on the web. Table 3.1 (from [32]) gives an overview of the artefacts created during those design steps. To enhance the separation of concerns, SHDM (OOHDM) keeps a strict separation between conceptual and navigational design.

The most important steps of SHDM are discussed in the following Sections.

[14]Universidade Católica de Brasíia, see `http://www.puc-rio.br/`

Steps	Artefacts
Requirements Gathering	Scenarios: User Interaction Diagrams:Design Patterns
Conceptual Design	SHDM Conceptual Model, composed by: • SHDM Conceptual Schema; • SHDM Conceptual Ontology; • Instances.
Navigational Design	SHDM Navigational Model, composed by: • SHDM Navigational Class Schema; • SHDM Navigational Context Schema; • Specification Cards describing: Contexts, Access Structures and Facets; • SHDM Navigational Ontology.
Abstract Interface Design	Abstract Data Views: Configuration Diagrams: ADV-Charts: Design Patterns
Implementation	Running application using the previous artefacts and the mechanisms supported by the target environment (parser, inference engine, Java classes, .jsp pages, etc.)

Table 3.1: An overview of the artefacts created by SHDM

3.4.1 Conceptual Design

The Conceptual Model(CM) is built during the Conceptual Design step. The CM shows classes and their relationships specifically related to a domain. Classes are described as in object-oriented (OO) UML models. Figure 3.20 (taken from [33]) gives an example CM for an academic department.
The conceptual model from the UML class diagram is then mapped to a DAML+OIL[15]

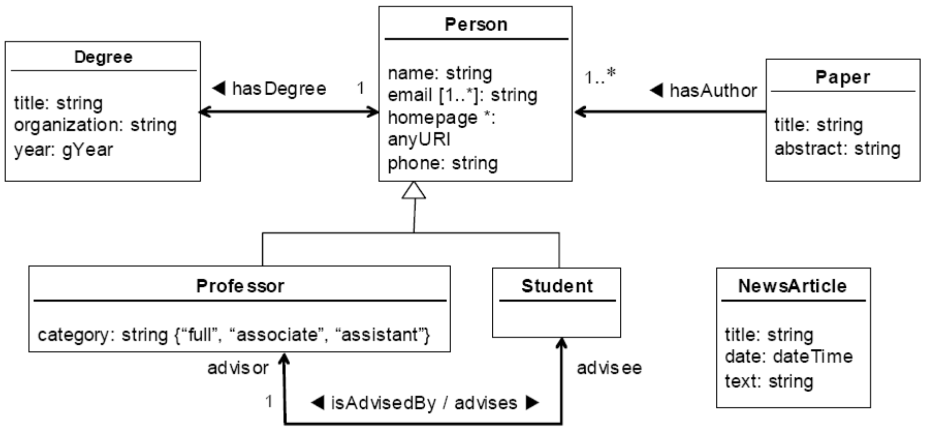


Figure 3.20: A SHDM Conceptual Model.

serialization format.

3.4.2 Navigational Design

The Navigation Design creates two models, the Navigation Class Model (NClM) and the Navigational Context Model (NCoM). The NClM defines all navigable objects as views over the application domain (”What” information can be reached). The NCoM defines the navigational contexts, their access structures and the links among them (”How” the information can be reached) [32].

3.4.2.1 Navigational Class Model

Figure 3.21 (from [33]) gives an example of a NClM for the example of Section 3.4.1.

The navigational classes use the same graphical representations as OOHDM.

[15]DAML+OIL is an OWL predecessor, see `http://www.w3.org/Submission/2001/12/`.

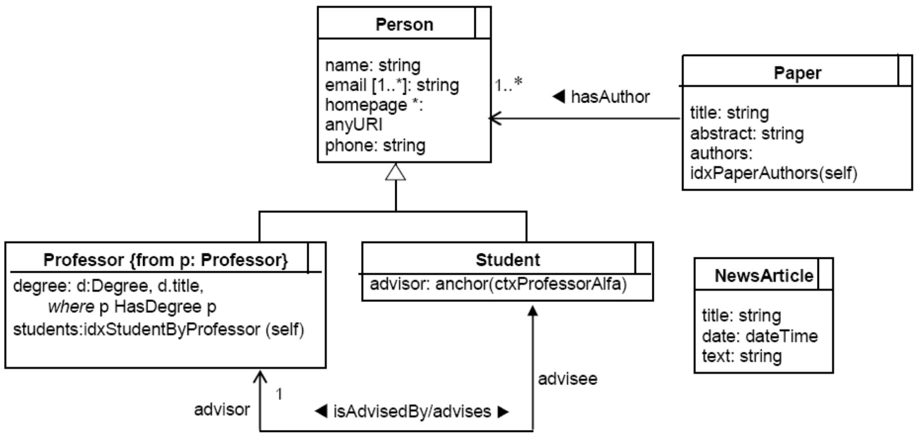


Figure 3.21: A SHDM Navigational Class Model.

These classes correspond to nodes and the links among them correspond to navigational links. Navigational classes represent views of conceptual classes, including directly mapped conceptual attributes, derived attributes and attributes from other conceptual classes [32].
Meaning:

directly mapped conceptual attributes These attributes are directly imported from the classes in the CM.

derived attributes These attributes can be derived (e.g., calculated) from attributes in the CM.

attributes from other conceptual classes These attributes are added to the NClM classes from other classes in the CM.

The mappings between the NClM and CM are specified using RQL[16] queries on the data in the CM. These queries are described in the attributes of the navigational classes (e.g., `select y from { Person } name { y }` for the `name` attribute of the `Person` navigational class)

3.4.2.2 Navigational Context Model

The NCoM is complementary to the NClM, and it allows the description of two important aspects [32]:

- The different ways how context objects can be grouped during navigation.

[16]RDQL - A Query Language for RDF, see `http://www.w3.org/Submission/RDQL/`

- The access structures to reach these context objects.

These context objects refer to navigational classes and define *how* the information can be browsed by the user in some context. The access structures are collections of links that provide access to the navigational objects in some context.

Figure 3.22 (from [33]) gives an example of the NCoM for the NClM from

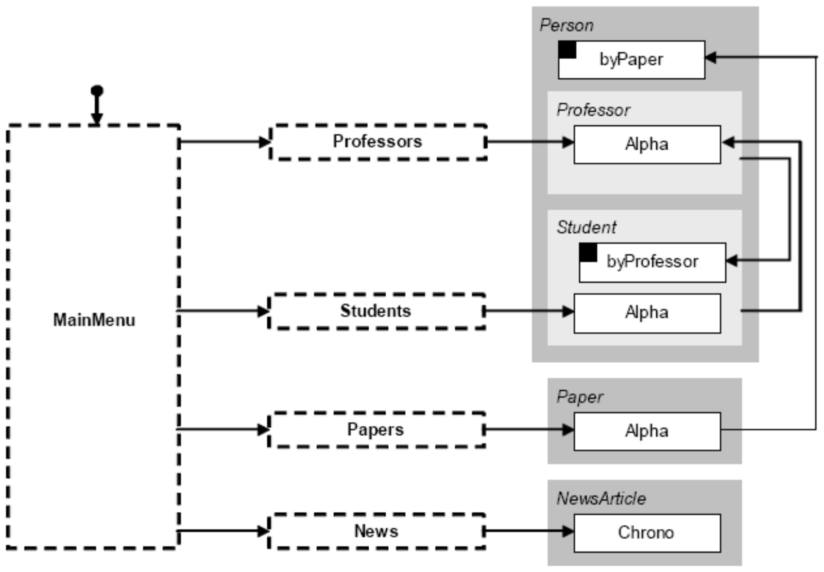


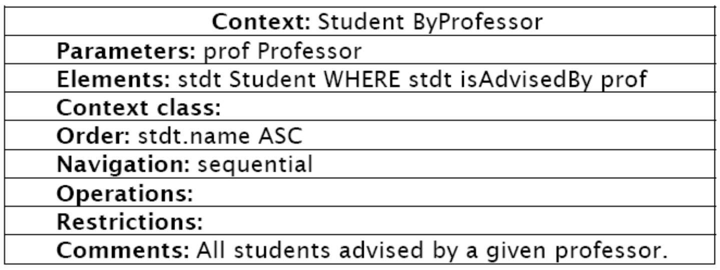
Figure 3.22: A SHDM Navigational Context Model.

Section 3.4.2.1. The context objects present in this example are `Professor`, `Student`, `Paper` and `NewsArticle`. `Student`, for example can be reached through the access structures `MainMenu` and `Students` and browsed by the access patterns alphabetical (`Alpha`) or by professor (`byProfessor`). The definition of a context is given by its Context Definition Card (CDC) [33].

Figure 3.23 (taken from [33]) shows such a CDC for the Student ByProfessor context. Notice the query expression defining the members of the context. The navigation for this context is sequential, with the Students shown in alphabetical order on name (`stdt.name`).

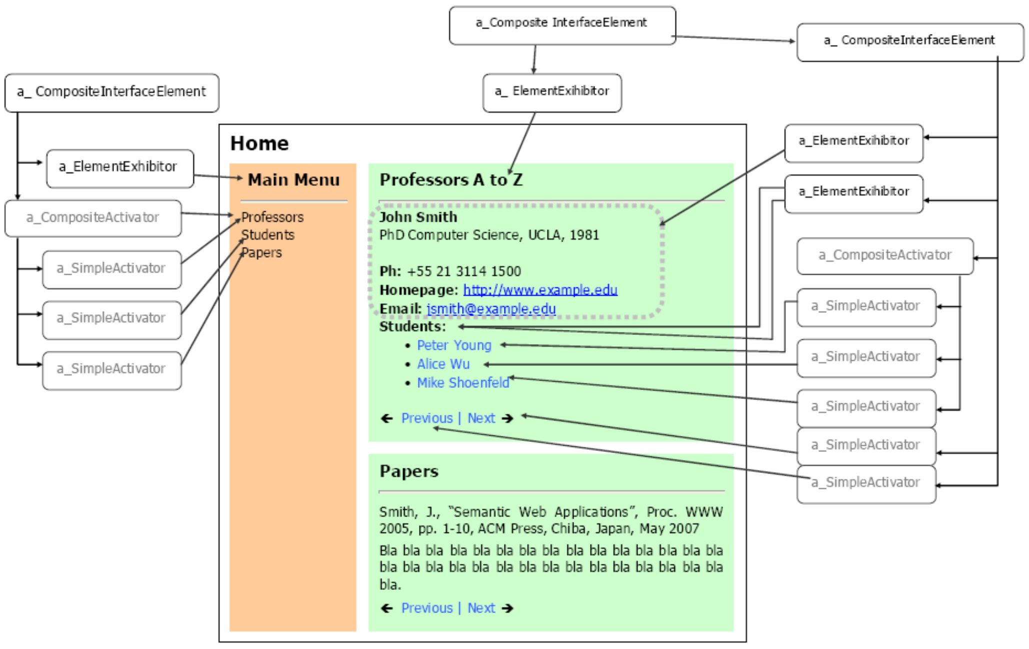
3.4.3 Interface Design

Abstract Interface Design focuses on making Navigation objects and application functionality available to the user at the abstract level. It allows modeling the information exchange between the application and the user. Separating the abstract and the concrete level of the Interface design allows the shielding of a significant



Context: Student ByProfessor
Parameters: prof Professor
Elements: stdt Student WHERE stdt isAdvisedBy prof
Context class:
Order: stdt.name ASC
Navigation: sequential
Operations:
Restrictions:
Comments: All students advised by a given professor.

Figure 3.23: A SHDM Context Definition Card for the `byProfessor` access pattern for the `Student` context.



John Smith
PhD Computer Science, UCLA, 1981

Ph: +55 21 3114 1500
Homepage: http://www.example.edu
Email: jsmith@example.edu
Students:
- Peter Young
- Alice Wu
- Mike Shoenfeld

← Previous | Next →

Smith, J., "Semantic Web Applications", Proc. WWW 2005, pp. 1-10, ACM Press, Chiba, Japan, May 2007
Bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla.
← Previous | Next →

Figure 3.24: A SHDM Abstract Interface .

part of the interaction design from platform evolution, as well as from the need to support users in various hardware and software runtime environments.
The Abstract interface uses the Abstract Widget Ontology. Figure 3.24 from [33] gives an example of an abstract interface and its widgets.

The Abstract interface widgets must be mapped onto Concrete Interface Widgets in order to be visible in the actual interface. For example, the abstract `SimpleActivator` widget from Figure 3.24 is mapped to a `Link` concrete interface widget. The mapping of the abstract onto the concrete widget ontology will record the actual interface elements chosen by the designer.

3.5 Conclusion

All of the different Ontology Based Website Design Methods presented in the previous Sections, provide a way to model the grouping of resources into webpages:

WSDM The content is described using Object Chunks. These Object Chunks are encapsulated in Nodes, linked together in the Navigational Model. The Nodes of the Navigational Model are then grouped onto webpages by the Site Structure Model.

Hera The Concepts of the Conceptual Model are contained in Slices from the Application Model. Instances for the top-level Slices then correspond to the webpages presented to the user.

OntoWeaver The Concepts from the Domain Ontology for the website are mapped to Page Nodes in the Site Structure Model. Each of these Page Nodes can be instantiated as a webpage.

SHDM The Classes of the Conceptual Model are extended with Navigational Links by the Navigational Class Model. This model forms the basis for the Navigational Context model which can be instantiated into a webpage.

All these models are captured using ontologies and thus all are the subject of ontology evolution (see Section 4). Since the subject of this thesis makes use of ontology evolution to keep track of the changes made to the structure of the website, each of these Website Design Methods would be suitable for the proposed method.

Because of the explicit formalization using ontology-languages of the different steps and the automated implementation generation possibilities, we have decided to use the WSDM Website Design Method as a basis for the research subject of this thesis. The different ontologies used will be described in Section 5, providing the reader with a sufficient basis for understanding the remainder of this thesis.

Chapter 4

Ontology Evolution

In an open and dynamic environment like the Semantic Web [18], the domain knowledge is continually evolving. This evolving knowledge means that the ontologies capturing this knowledge need to evolve as well. This is where ontology evolution comes into play.

Ontology evolution can be defined as the adaptation of an ontology to the arisen changes and the consistent management of these changes. This is not a trivial process, due to the variety of sources and consequences of changes and thus cannot be performed manually by an ontology engineer. Because of this, the ontology evolution process needs to be supported by an ontology evolution approach.

The following Sections will describe three such approaches, followed by a conclusion.

4.1 SIKS Ontology Evolution Approach

The Dutch Research School for Information and Knowledge Systems (SIKS)[1] supports this ontology evolution approach developed by Michel Klein [43] at the Vrije Universiteit Amsterdam[2].

This approach is centered around *ontology changes*. An ontology change is defined as an action on an ontology that results in an ontology that is different from the original version.

[1]The Dutch Research School for Information and Knowledge Systems, see `http://www.siks.nl/`

[2]Studie Informatica, Vrije Universiteit Amsterdam, see `http://www.cs.vu.nl/`

The following Sections describe the SIKS Ontology Evolution Approach. Section 4.1.1 describes the way this approach represents changes. Section 4.1.2 describes the way changes are detected in the SIKS Ontology Evolution Approach. And finally, Section 4.1.3 describes how the change detection and change representation from the previous Sections is used by the change handling process of the approach.

4.1.1 Change Representation

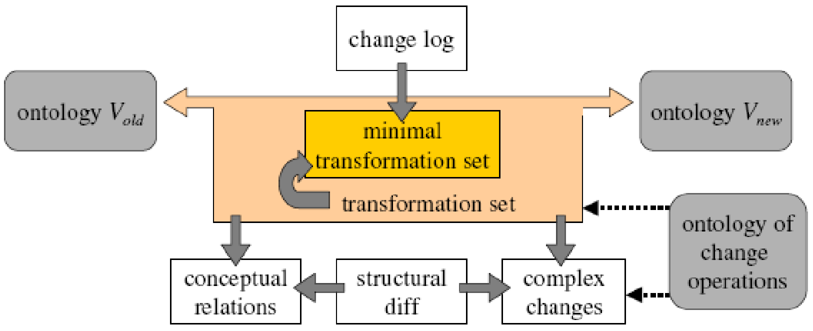


Figure 4.1: The relationships among the Change Representations and the Operations from the Ontology of Change Operations (taken from [43])

Figure 4.1 gives an overview of the different change representations used by this ontology evolution approach, together with their transformations. Each of these change representations provides different information at a different level of detail. Starting with two versions V_{old} and V_{new} of an ontology, the possible forms of representation are:

- Ontologies only: the old version V_{old} and the new version V_{new} of the ontology provide no explicit change information, but can be used as a basis to find other change information;
- Log of changes: a record of the changes as they are performed; a list of changes that applied to V_{old} results in V_{new} ;
- Structural diff: a mapping between concepts and properties in one version and their counterparts in the new version, together with a list of added and removed concepts;
- Conceptual relations: an explicit specification of the conceptual relations between concepts in V_{old} and corresponding concepts in V_{new} .

The following Sections will now describe the main components of this ontology evolution approach [43].

4.1.1.1 A meta-ontology of change operations

A meta-ontology of basic change operations is directly related to the ontology language itself and constitutes a set of operations to build an ontology in this language. The ontology evolution approach uses the meta-model of the OWL and OKBC Ontology Languages. By defining for each of the elements of the meta-model ”add”, ”remove” and ”modify” operations, a set of operations is created that enables the description of all possible changes for the ontology language.

4.1.1.2 The notion of complex changes

Besides the basic change operations, the ontology of change operations of this ontology evolution approach also contains *complex change operations*. Complex operations provide a mechanism for grouping a number of basic operations that together constitute a logical entity.
Complex changes could also contain information about the implication of the operation on the logical model of the ontology. To identify complex changes, the logical theory of the ontology is needed, while the structure of the ontology is sufficient to identify the basic changes.

4.1.1.3 The notion of a transformation set

A transformation set provides a set of change operations that specify how V_{old} can be transformed into V_{new}. The transformation set uses the operations from the ontology of changes. A transformation set is not unique, there are often multiple ways to construct a transformation set for a specific change.
A *minimal* transformation set is a special variant of a transformation set. It consists of a set of operations that is *sufficient and necessary* to transform V_{old} into V_{new}.

4.1.1.4 A template for change specification

This is a template that can be used to describe how two ontology versions are related. Such a template has the following elements:

- Descriptive meta-data: this is book-keeping information which can be used to identify the versions and changes in a setting of collaborative development.

- Minimal transformation set: this transformation set can be used to re-execute the change, to translate or re-interpret data sets, and as a basis for deriving additional information about the change.
- Conceptual relations: this is the relation between concepts across versions as specified by the ontology engineer.
- Complex changes:Together with the minimal transformation set, the complex operations can be used to create data transformation scripts and to determine in more detail the effect of changes on data accessibility and specific logical queries.
- Change rationale: this is the intention (e.g. fix of an error, specification, update, ...) behind the change. The intention can be used to decide which version to use and can help to visualize the change.

4.1.2 Change Detection

According to the SIKS ontology evolution approach, change detection happens by *finding* or *deriving* changes.

4.1.2.1 Finding Changes

When only V_{old} and V_{new} are available, a transformation set and a structural diff can be created to find changes.

4.1.2.2 Deriving Changes

Change information that is already available can be used to derive additional information about the change.
The SIKS ontology evolution approach describes the following possibilities to derive changes:

Change log → minimal transformation set Many ontology-editing tools provide logs of changes, these can be transformed into transformation sets by translating the operations into the vocabulary of basic changes and removing all redundant changes.

Transformation set → minimal transformation set The transformation set can be transformed into minimal transformation sets by removing all redundant changes.

Transformation set $\rightarrow$ complex changes A transformation set consisting of basic operations, can be extended with heuristics to combine the basic operations into complex change operations.

Structural diff $\rightarrow$ complex changes A structural diff can be used to create more useful change descriptions.

Transformation set $\rightarrow$ conceptual relations A transformation set with both basic and complex operations defined between versions, can be extended by heuristics to suggest conceptual relations between frames in versions to the user.

Structural diff $\rightarrow$ conceptual relations Conceptual relations can be derived from a structural diff.

V_{old}, V_{new} and structural diff $\rightarrow$ conceptual relations An automated reasoner can be used to derive conceptual relations between the concepts in the old and new version of the ontology.

4.1.3 The Change Process

This process defines how the models described in the previous Sections can be used. Figure 4.2 (taken from [43]) gives an overview of this process. The process starts off by determining the specific versions from the ontology's life trace that are relevant for the ontology related task in question.
After this, change information is iteratively generated, until enough information is available to perform the task at hand. This change information is called a *Change Specification* and the SIKS ontology evolution approach specifies the following methods to build this Change Specification [43]:

Generating a Transformation Set This can be done either by analyzing a log of an ontology editor or by comparing two ontology versions.

Generating Two Versions This can be done by applying the transformation set to one version.

Generating Complex Changes This can be done from editor logs or by applying rules or heuristics to the transformation set.

Generating Evolution Relations These relations provide a mapping between corresponding constructs in the old version and the constructs in the new version of the ontology, specifying which concept has evolved in what other concept. If the ontology contains persistent identifiers for concepts and relations, those can be used to generate the evolution relations. Alternatively,

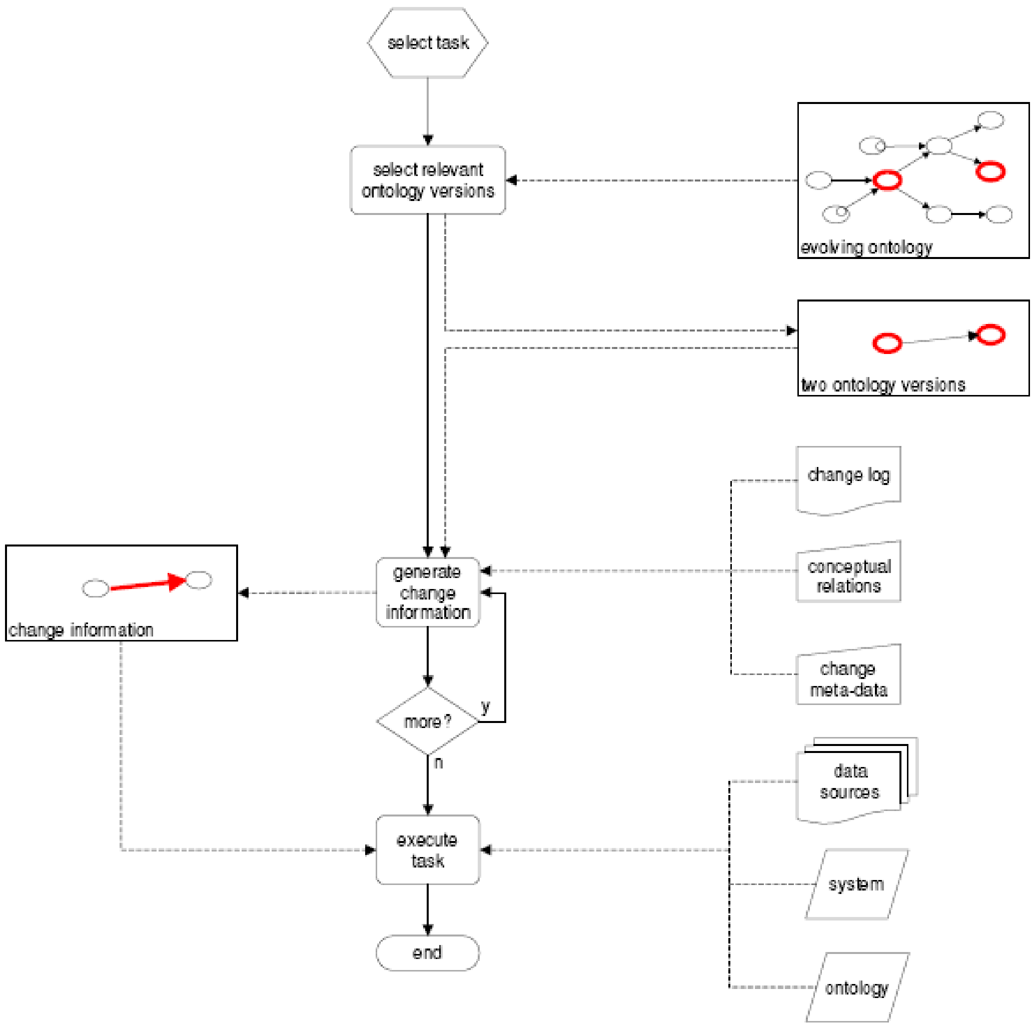


Figure 4.2: The model for managing the ontology change process for ontology-related tasks

a change log can be used to find the mappings.If a log is not available either, then mapping heuristics have to be applied.

Generating Conceptual Relations The conceptual relation between two concepts specifies the intended semantic relation between the two versions of a concept or relation. If both versions of the ontology are available and the semantics of the ontology can be expressed in a Description Logic the subsumption and equivalence relations according to the definitions can be computed automatically by reasoning systems. If an exact computation of the logical relation can not be done, heuristics can be applied to the transformation set to suggest conceptual relations to the human expert that understands the domain of discourse.

4.2 KAON Ontology Evolution Approach

The Karlsruhe Ontology and Semantic Web framework (KAON) has been developed at the University of Karlsruhe[3] and provides a platform needed to apply Semantic Web technologies to E-commerce and B2B scenarios, knowledge management, automatic generation of Web portals, E-Learning, E-Government etc. The KAON framework uses the KAON ontology language, an RDF based language with many additions and changes to the standard. The KAON framework includes an ontology evolution approach developed by Ljiljana Stojanovic [42].

This ontology evolution approach manages changes in six steps and is illustrated by Figure 4.3. These six phases are [42]:

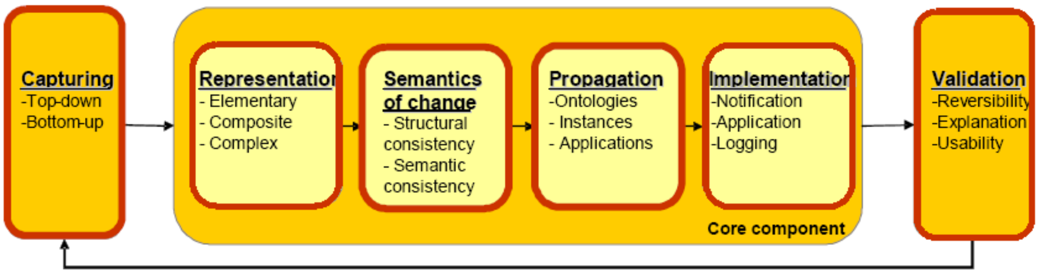


Figure 4.3: The Ontology Evolution Process (taken from [42]).

1. The process starts with capturing changes either from explicit requirements or from the result of change discovery methods;

2. Next, the changes are represented formally and explicitly as one or more ontology changes;

3. Then, the semantics of change phase prevents inconsistencies by computing additional changes that guarantee the transition of the ontology into another consistent state;

4. In the change propagation phase, all *artefacts* (applications, other ontologies, etc) that depend on the ontology are updated;

[3]Universität Karlsruhe, see `http://www.uni-karlsruhe.de/`

5. During the change implementation phase, the required and induced changes are applied to the ontology in a transactional manner;

6. In the change validation phase, the user evaluates the results and restarts the cycle if necessary.

The *core* ontology evolution process are the four middle phases from Figure 4.3. These phases are described in the following Sections.

4.2.1 Change Representation

The role of the change representation phase of the KAON ontology evolution approach is to map a request for a change into one or more ontology changes. To represent changes, three levels of abstractions of ontology changes are are introduced into the KAON language. These levels are described in Figure 4.4 (from [42]) The three levels of abstraction are:

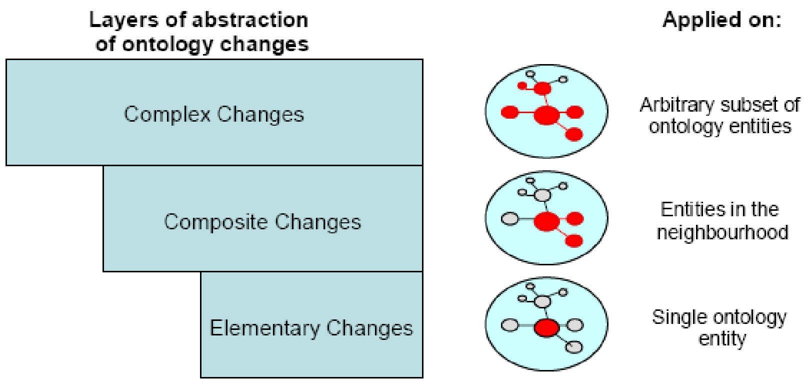

Layers of abstraction of ontology changes
Applied on:
Complex Changes
Composite Changes
Elementary Changes
Arbitrary subset of ontology entities
Entities in the neighbourhood
Single ontology entity

Figure 4.4: The different layers of abstraction for ontology changes and their applicability.

An elementary ontology change This is an ontology change that modifies (adds or removes) only one entity of the ontology model. It can be seen as an isolated modification of an ontology;

A composite change This is an ontology change that modifies (creates, removes or changes) the neighbourhood of an ontology entity. Composite changes specify coarse-grained changes. They are more powerful since an ontology engineer does not need to go through every step of the sequence of basic changes to achieve the desired effect;

A complex change This is an ontology change that can be decomposed into any combination of at least two elementary and composite ontology changes. Complex changes encompass all ”real-life” changes not included in the elementary and composite changes. Complex changes raise the level of abstraction and reusability even further.

The above mentioned changes are represented as instances of an *evolution ontology* . This is an ontology which explicitly represents semantic information about ontology entities, changes in the ontology and mechanisms to discover and resolve changes.

4.2.2 Semantics of Change

The application of an elementary change to an ontology can induce inconsistencies in other parts of the ontology. The task of the semantics of change phase is to enable the resolution of induced changes in a systematic manner, ensuring consistency of the whole ontology.

The KAON ontology evolution approach distinguishes structural and semantic inconsistency. Structural inconsistency arises when the *constraints of the ontology model* are invalidated. Semantic inconsistency arises when the *meaning* of an ontology entity is changed due to the changes performed in the ontology.

4.2.2.1 Semantic inconsistency resolution

To allow for semantic inconsistency resolution, the standard ontology model has to be enriched with semantic information that exactly characterizes the concept's semantic properties and expected ambiguities, including the properties that are prototypical for a concept and that are exceptional or essential as well as the behavior of a properties over time and the degree of applicability of properties to subconcepts.

4.2.2.2 Structural inconsistency resolution

Since an ontology engineer can easily overlook some part of the overall modification, it cannot be expected that the generation of additional changes needed to keep the consistency can be done manually, so two main approaches were adopted from the database community to ensure the consistency [42]:

1. The Procedural approach
   This approach resembles classical problems of schema change:
   specify the semantics of ontology changes (i.e. preconditions and postconditions) and accordingly, specify rules to preserve the consistency of the

resulting ontology. This approach is extended by specifying a multiple set of rules (i.e. the evolution strategies) for ensuring the consistency. And, the so-called metarules (i.e. the advanced evolution strategies) are defined for controlling the set of consistency enforcing rules that have to be used.

2. The Declarative approach
   This approach models the ontology evolution as reconfiguration-design problem solving task, such that the problem is reduced to a graph search where the nodes are evolving ontologies and the edges represent the changes that transform the source node into a target one. The search of the graph is guided by the constraints provided partially by an ontology engineer and partially by a set of rules defining the ontology consistency.

4.2.3 Change Propagation

The task of the change propagation phase of the KAON ontology evolution approach is to automatically bring all dependent artefacts into a consistent state after an ontology update has been performed.

The used approach depends on the artefact type:

4.2.3.1 The effect of changes on the Dependent Ontologies

This problem can be solved by recursively applying the ontology evolution process to the dependent ontologies.

4.2.3.2 The effect of changes on the Ontology Instances

When the ontology is modified, ontology instances need to be changed to preserve consistency with the ontology.
Two types of evolution can be distinguished:

1. Metadata evolution:
   The instances are distributed over the Web.
   The resolution of this case can be performed in three steps:
   (a) The instances that may depend on a change have to be gathered into a temporary ontology. The output of this step is a list of instances together with a reference to the web documents containing them;
   (b) The temporary ontology is a dependent ontology consisting of only ontology instances. It must be transformed to conform to the modified

ontology. This step provides output in the form of a list of modified instances with a reference to the corresponding web resource;

   (c) In the last step, the ”out-of-date” instances on the Web are replaced with corresponding ”up-to-date” instances.

2. Knowledge base evolution:
   In this case, the instances are organized in an instance pool. These instances are then transformed to conform to the modified ontology.

4.2.3.3 The effect of changes on the Applications

When an ontology is changed, applications based on the changed ontology may no longer work correctly.
An application can be maintained semi-automatically only if there is metadata describing which ontology and/or which ontology entities that application uses. In order to avoid overhead, the ontology changes are categorized in changes that require a modification of the application and changes that do not require a modification of the application.

4.2.4 Change Implementation

The role of the change implementation phase of the KAON ontology evolution approach is threefold [42]:

1. To inform an ontology engineer about all consequences of a change request;

2. To apply all the (required and derived) changes;

3. To keep track of the performed changes.

These three roles will be described in the following Sections, followed by a description of the evolution ontology and the Evolution Log used by this approach.

4.2.4.1 Notification of the Consequences of a Change

To avoid performing undesired changes a list of all implications to the ontology and dependent artefacts should be generated and presented to the ontology engineer who modifies the ontology before applying a change to an ontology. When the changes are approved, they are performed by successively resolving changes from the list. If changes are canceled, the ontology should remain intact.

4.2.4.2 Change Application

During this sub phase all changes (i.e. required and derived changes) are applied to a consistent ontology and result into a new consistent state of this ontology. All these changes act like an atomic transaction, although the changes are executed step by step.

4.2.4.3 Change Logging

The last sub phase of the change implementation phase needs to keep track of the performed changes.
The KAON ontology evolution approach makes use of an evolution ontology and the Evolution Log. The evolution ontology is a model of ontology changes enabling better management of these changes. The Evolution Log tracks the history of applied ontology changes as an ordered sequence of information (defined through the evolution ontology) about a particular change. Figure 4.5 illustrates the dependencies between the Evolution Ontology and Log and the Domain Ontology.

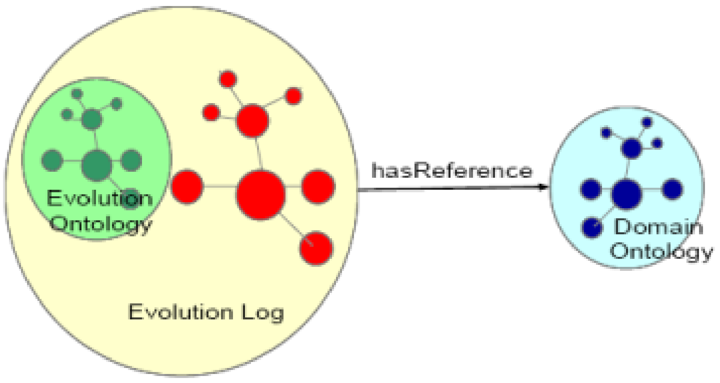


Figure 4.5: The dependencies between the Evolution Ontology, the Domain Ontology and the Version Log.

4.2.4.4 Evolution Ontology

The Evolution Ontology of the KAON ontology evolution approach is a meta-ontology that is used as a backbone for creating Evolution Logs. Figure 4.6 shows part of this evolution ontology. It models what changes, why, when, by whom and how they are performed in an ontology. Therefore, the most important concept is the `Change` concept. The structure of the hierarchy of ontology changes reflects the underlying ontology model by including all possible types of changes.

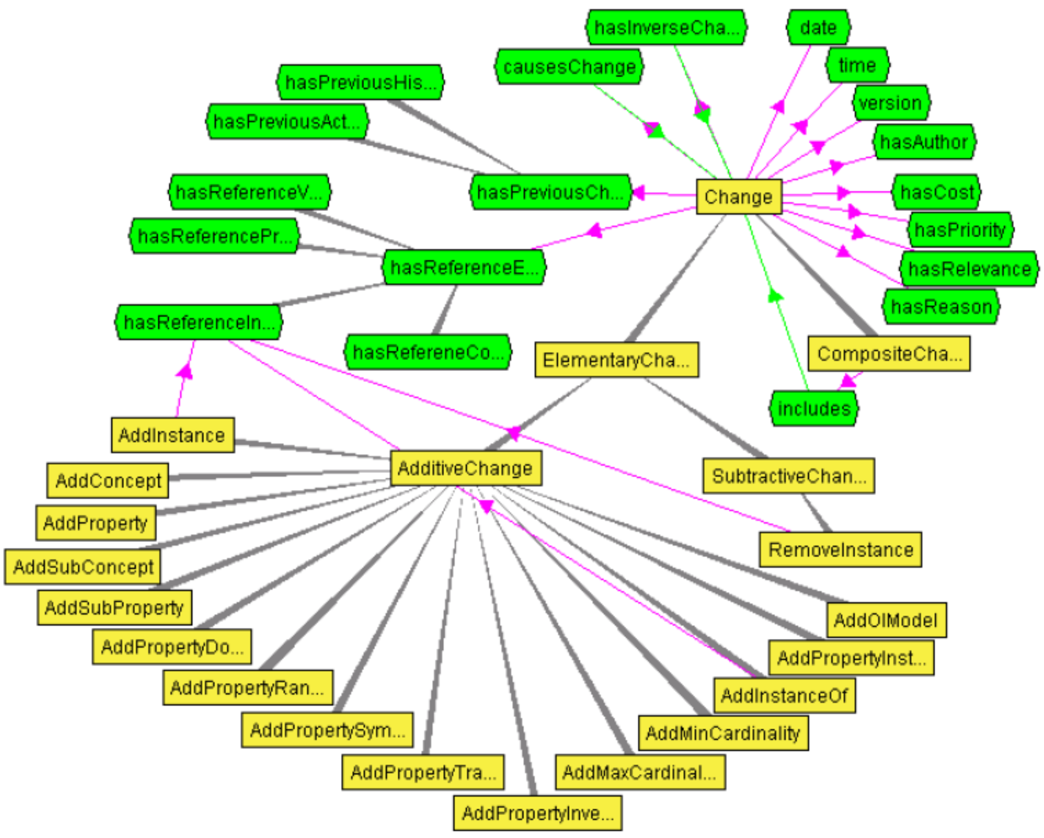


Figure 4.6: Part of the Evolution Ontology.

The Evolution Ontology allows the association of additional information to changes (e.g. date and time of change, version number, priority, cost ...), but also the expression of dependencies. The dependencies can run among the changes (e.g. the `causesChange` property, the `hasPreviousChange` property) or between the changes and the concepts of the domain ontology (e.g. the `hasReferenceEntity` property).

4.2.4.5 Evolution Log

The Evolution Log records an exact sequence of changes that occurred when an ontology engineer updated an ontology. Therefore, it contains instances of subconcepts of the concept `Change`, which include the elementary as well as the composite ontology changes. Each instance contains data about a particular change. For example, all changes in the log have a timestamp, author, version, etc.

4.3 WISE Ontology Evolution Approach

The WISE ontology evolution approach has been developed by Peter Plessers [13], member of the WISE research group[4] at the Vrije Universiteit Brussel[5].

This ontology evolution approach has resulted in a framework that works through the usage of a *Version Log*. Such a Version Log stores the different versions a concept defined in the ontology passes through during its life cycle. Storing these versions happens by using the *Version Ontology* as described in Section 4.3.8. The Version Log allows the definition of changes as queries on the Version Log, using the *Change Definition Language* as described in Section 4.3.9. The execution of these queries results in a listing of the applied changes, called an *Evolution Log*. The definitions for these changes can vary, each interested party can define its own set of changes, suited to meet certain needs.

The framework consists of seven phases used in two different contexts. Figure 4.7 gives an overview of these phases and illustrates the different contexts. The

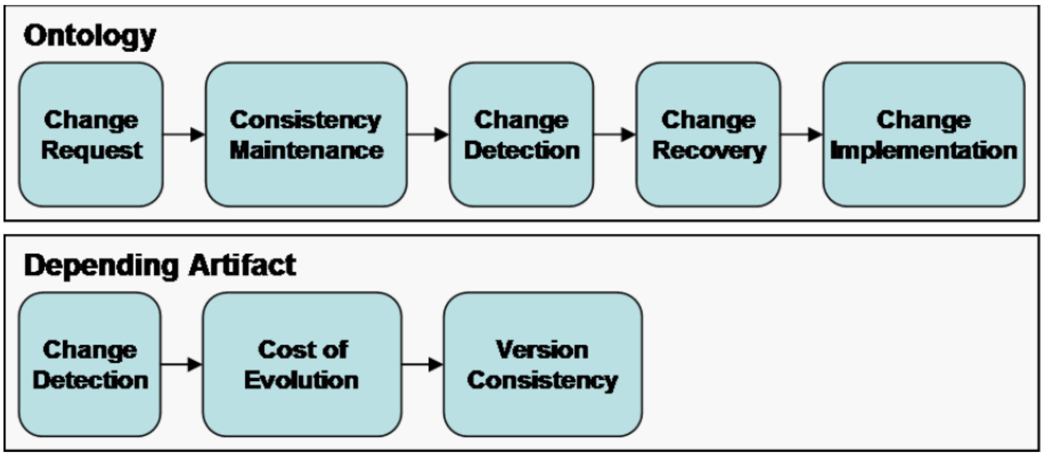


Figure 4.7: The seven phases of the Ontology Evolution Framework.

top part of Figure 4.7 contains the phases applicable in the context of the evolution of the ontology itself, while the bottom part of the Figure contains the phases relevant in the context of the evolution of the depending artifacts (the ontologies or applications depending of the evolving ontology).

[4]Web & Information Systems Engineering, see `http://wise.vub.ac.be/`
[5]Vrije Universiteit Brussel, see `http://www.vub.ac.be`

The following Sections specify the seven phases of the WISE ontology evolution framework, followed by Section 4.3.8 describing the ontology used by the Version Log and Section 4.3.9 describing the language used to define changes.

4.3.1 Change Request

This phase gives the ontology engineers the ability to specify their request for change through predefined changes.
Different types of changes are distinguished by the ontology evolution framework:

Primitive Changes Primitive Changes modify exactly one element of the ontology and cannot be decomposed any further. Primitive Changes are in turn divided into domain independent and domain dependent changes. The set of domain dependent changes are defined for a particular domain. The set of domain independent changes is derived from the underlying ontology language and expressed as a set of modifications to the constructs of the ontology language.

Composite Changes This type of changes modify more than one element of the ontology and offer a higher level of abstraction.

Meta Changes This type of changes specify the implications of a change, they provide information about a change.

The changes requested by the change request cause pending versions for concepts in the Version Log. These pending versions represent the outcome of requested changes and are not yet implemented in the actual ontology or checked for consistency.

4.3.2 Consistency Maintenance

This phase fulfills three tasks: checking the consistency of the ontology, resolving inconsistencies and checking the backward compatibility of the changed ontology.

The first task, consistency checking, is solved through the usage of a reasoner. First, the pending changes from the previous phase are applied to a copy of the ontology. Then, the reasoner is used to check the consistency of this copy.

To be able to resolve inconsistencies, the onology evolution framework extended the algorithm used by the reasoner to explicitly keep track of the internal axioms it uses. Keeping track of these axioms, gives a complete description of the inconsistencies detected. This complete description can then be used by the ontology engineer to select the actions necessary to resolve the inconsistencies. The ontology evolution framework expresses these actions through a set of rules. Such a rule calls another rule or adds a new, so-called *deduced change* to the Change Request.

The Cost of Evolution phase described in Section 4.3.6 determines the backward compatibility of the intermediate versions for depending artefacts. When a version of the ontology is backward compatible with another for a certain depending artefact, this means that the old version of the ontology can be replaced with the new version without breaking the depending artefact. Inconsistency can generally be resolved in several ways and determining whether a solution maintains backward compatibility or not, helps the ontology engineer choose from the possible solutions.

As a result of this phase, the Change Request has been updated to allow the ontology to evolve from one consistent state into another. The ontology engineer can then decide to accept or reject the proposed changes. When the changes are accepted, the pending versions in the Version Log are changed into confirmed versions. When the proposed changes are rejected by the ontology engineer, the pending versions are removed from the Version Log.

4.3.3 Change Detection

During this phase, changes are detected that were not specified during the Change Request phase, but that did occur as a consequence of the modifications to the ontology.

The automatic detection of changes has the following advantages:

- The same ontology modification can be achieved by composite changes of different granularity, the change detection mechanism allows the inclusion of all these complex changes in the Evolution Log.
- Meta Changes can be automatically detected. This allows the ontology engineer to focus on the essentials during the Change Request phase.
- Ontology engineers and the maintainers of the depending artefacts may use different definitions for the same change. Automatic change detection allows the generation of Evolution Logs using these different definitions.
- Some tools for creating Change Requests only provide a limited set of changes that can be specified, automatic change detection smoothens out these differences in tool support.

Since changes are defined as queries on the Version Log, change detection is simply the execution of these queries. The result of this phase is an Evolution Log containing both the changes from the change requests and the changes detected by the ontology evolution framework.

4.3.4 Change Recovery

An ontology engineer may have used a sequence of changes instead of the right complex change. After each step in that sequence of changes, the framework may add deduced changes to solve inconsistencies. When the sequence of changes is then considered as a whole, some of these deduced changes may be superfluous. During the Change Recovery phase, the framework should recover from these superfluous changes.

This recovery happens by undoing the versions in the Version Log that were added because of the deduced changes. The resulting temporary ontology is then checked for consistency. If the temporary ontology remains consistent, the framework recommends the removal of the relevant versions to the ontology engineer, otherwise the versions are restored and the deduced changes are not recovered.

4.3.5 Change Implementation

During all the previous phases, the changes have only been applied to the local copy of the ontology. The goal of the Change Implementation phase is to apply the changes to the public version of the ontology. This is simply done by replacing the original public version of the ontology with the changed, local copy of the ontology.

4.3.6 Cost of Evolution

In the Cost of Evolution phase, the framework determines which versions of the ontology remain backward compatible and allow updating without changing the depending artefacts. The framework also determines which parts of the dependent artefact need updating when the ontology is updated to a non-backward compatible version. Both these tasks are reached using the consistency checking and backward consistency checking tasks from Section 4.3.2.

Based on this information provided by the framework, the maintainers of the artefacts depending on the ontology should decide if they still agree with the changed ontology and if the changes warrant an update. If there is an intermediary backward compatible version of the ontology, the maintainers can choose to update to only to this intermediary version.

4.3.7 Version Consistency

Figure 4.8 gives an overview of the input provided by the framework to guide an update decision. The Change Detection phase describes the changes that have

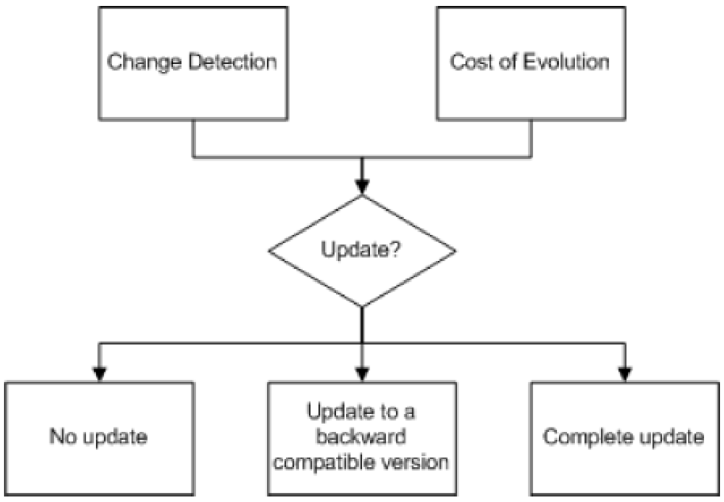


Figure 4.8: Factors guiding the decision to update.

occurred and the Cost of Evolution phase describes which intermediary versions are still backward compatible and which parts of the depending artefact are sensitive to the changes. To make sure that an updated depending ontology remains consistent with the ontology it depends on, the depending ontology starts a new iteration of the ontology evolution framework. The change request for this iteration of the depending ontology contains the deduced changes necessary to restore consistency with the ontology it depends on.

4.3.8 Version Ontology

For each class, property and individual that is created in the Domain Ontology, an associated EvolutionConcept instance from the Version Ontology is created in the Version Log. Such an EvolutionConcept instance keeps, besides a reference to the concept in the Source Ontology, a list of past and current versions of that concept. Whenever a change request for a concept in the ontology is executed, a new Version instance is added to the associated EvolutionConcept instance, representing the new version of the referred concept. Figure 4.9 (from [13]) gives an overview of the Version Ontology.

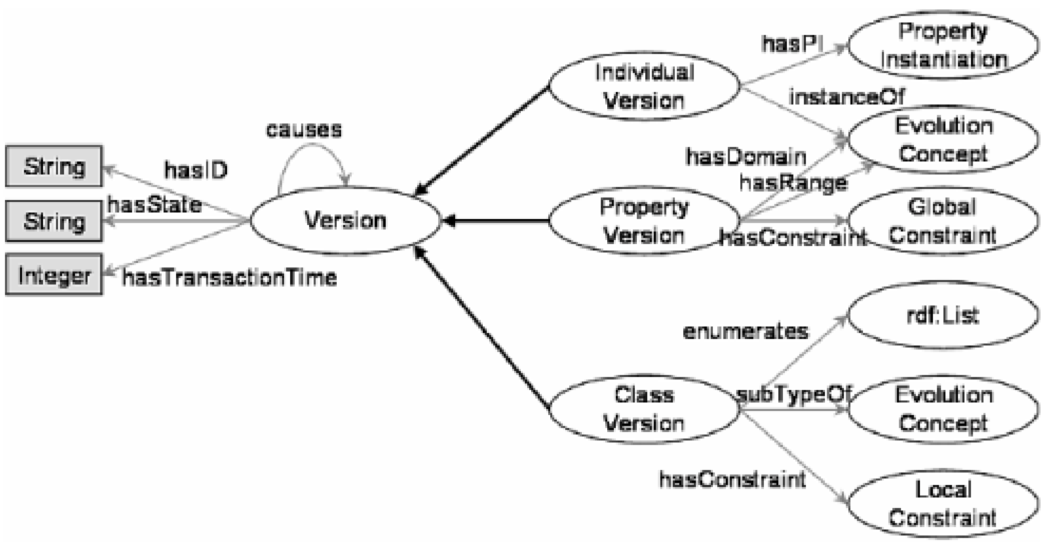


Figure 4.9: An abbreviated Version Ontology taken from [13].

4.3.9 Change Definition Language

The Version Log of the WISE ontology evolution approach uses an explicit timeline for the different versions. This explicit timeline aspect provides the possibility to check properties of past versions of ontology concepts by means of conditions. These conditions are used to formulate definitions for the different types of changes as described in Section 4.3.3. The conditions are resolved by pattern matching. The syntax of the conditions allows the querying of the Version Log, to determine the presence or absence of a property with a certain target for a certain version of a source.

4.4 Conclusion

Because all of the ontology evolution approaches presented in the previous Section provide a way to capture the modifications performed on an ontology and each of the ontology evolution approaches contains a way to define and detect these changes, namely:

- The SIKS Ontology Evolution Approach uses the Change Specification Language [43], based on the meta-ontology of change operations from Section 4.1.1.1;

- The KAON Ontology Evolution Approach uses the evolution ontology (see Section 4.2.4.5) in combination with constraints to specify changes. A declaritive language has been suggested but not yet defined [42];
- The WISE Ontology Evolution Approach uses the Change Definition Language described in Section 4.3.9.

We could choose any of these approaches as applicable for the subject of this thesis.

In the light of our earlier personal experience with an extended version of the WISE ontology evolution approach, we have decided to use this approach to track the changes to the ontologies that make up the higher-level structure of the website.

Chapter 5

WSDM Ontology

This Chapter provides an overview of the ontology used by the WSDM design methodology. As described in Section 3.1 on Page 16, WSDM uses an OWL ontology to formally represent the information and functionality modeling and to make the semantics of the different WSDM design models explicit. The internal use of the WSDM ontology allows the following [40]:

Generate content- and functionality annotations: The explicit semantic information available from the conceptual information and functional modeling is used to automatically generate semantic annotations about content and functionality.

Generate structural annotations: The explicit semantics from the high level conceptual modeling primitives of the WSDM ontology allow the generation of semantic annotations about the structure, organization and presentation of the website.

Allow interoperability between design methods: The formally defined design models make it possible to import from or export to other web design methods.

The following Sections describe the major modeling concepts and different metamodels of the WSDM ontology needed by the Transformation Pipeline (see Section 3.1.5 on Page 21). We start by describing the Object Chunk concepts, followed by the description of the Navigational, Site Structure, Presentation, Behavior, Style & Template and Page model concepts.

5.1 Object Chunks

As described in Section 3.1.3.1.2 on Page 19, an object chunk is a data model describing the necessary information and functionality needed to perform a task from the Task Modeling phase (see Section 3.1.3.1.1, p. 18). Since OWL is used by WSDM as the internal modeling language, Object Chunks are collections of OWL Statements (see Section 2.2.2). Using OWL allows importing from or linking to existing (OWL) ontologies. Figure 5.1 (from [40]) describes the concepts needed. An Object Chunk is described by an `ObjectChunk` concept which is composed of statements (described by the `Statement` concept) which is in turn composed of Classes (described by the `Class` concept) connected to each other by a Object Properties.

The `ObjectChunkFunction` concept from Figure 5.1 allows the attachment

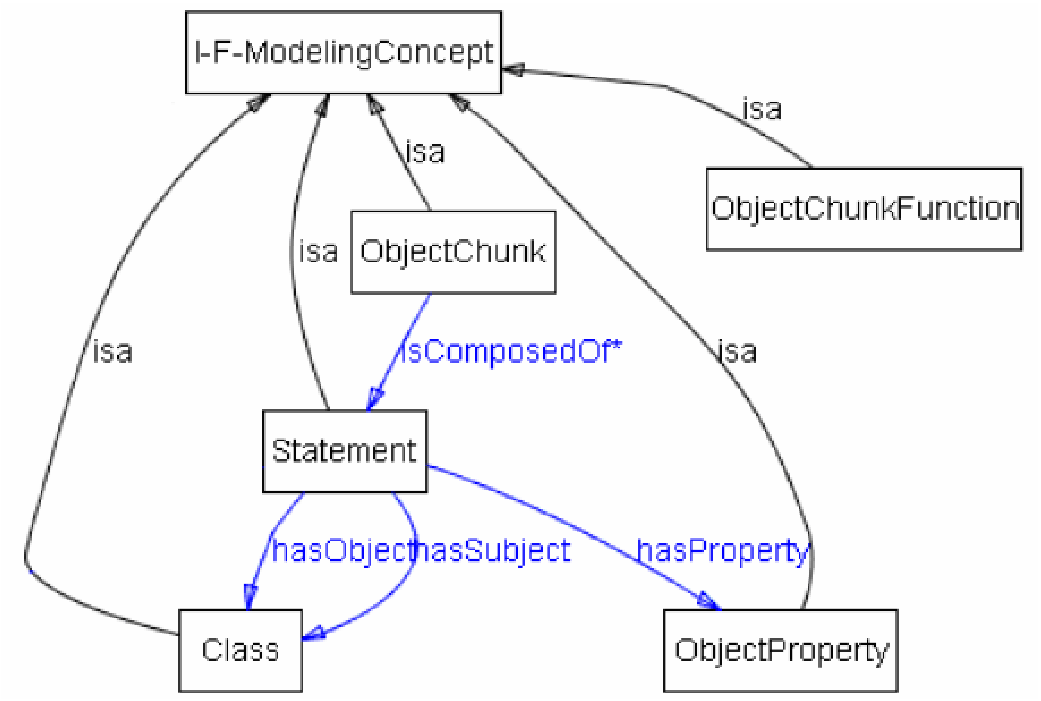


Figure 5.1: WSDM Information & Functional Modeling Concepts.

of functionality to `ObjectChunk` concepts. Figure 5.2 gives an overview of the Object Chunk Functions. These functions allow the *System* to *create* or *remove* instances of the associated concepts and they allow the *User* to *upload* a file or *fillout* a value for the associated concept or *select* an instance from the associated concept.

Figure 5.1 lacks a way of expressing DataTypeProperty concepts, since the standard OWL DataTypes are not sufficient to express the multimedia types used in web applications. These multimedia DataTypeProperties are expressed by the

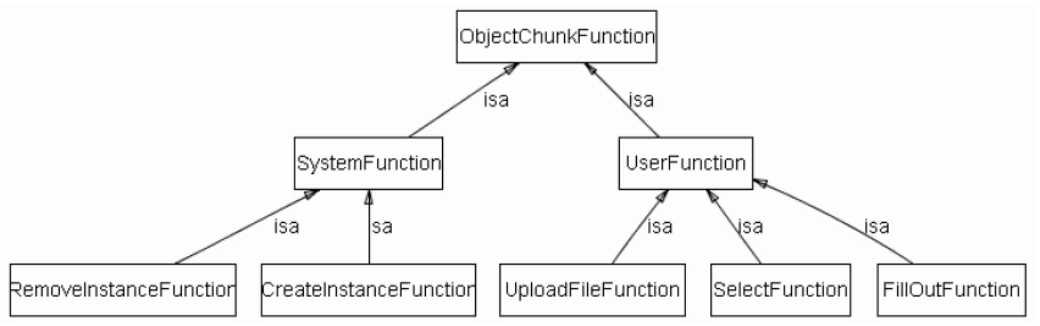


Figure 5.2: WSDM Object Chunk Functions.

`MultimediaConcept` concept, a subconcept of the `Class` concept. DataType-Properties are then modeled through an `ObjectProperty` between a Class and such a `MultimediaConcept`. The `MultimediaConcept` has in turn a number of subconcepts, representing the most common primitive multimedia types found in web applications: `String`, `Integer`, `Image`, `Email`, `Audio`, `Video` and `Resource`.

5.2 Navigational Model

The navigational model concepts are divided in `Nodes` and `Links` in between these nodes. An overview of this is given in Figure 5.3 (from [40]).

A `RootNode` is a special kind of `Node` defining the root of a navigational

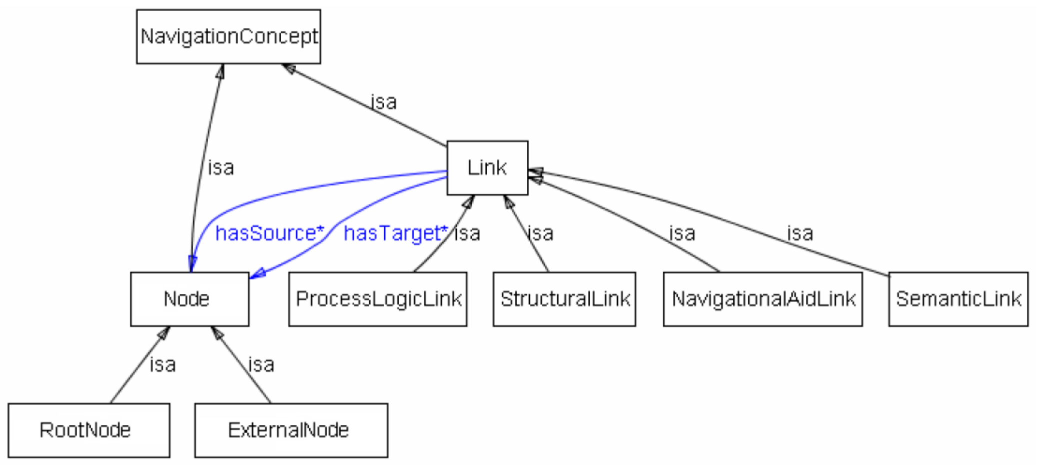


Figure 5.3: WSDM Navigational Model.

track (see Section 3.1.3.2 on Page 19) and an `ExternalNode` refers to external resources (e.g., pages from an external website). The Object Chunks from the previous Section are connected to the internal nodes, to allow different contexts for one Object Chunk.

Four subtypes of the `Link` concept are defined [40]:

Process logic link Such a link expresses part of a workflow (e.g., the checkout procedure for an e-commerce website's shopping cart):

Structural link Such a link is used to create organizational structures in information or functionality (e.g., linear, hierarchical, web, etc. structure):

Navigation aid link Such a link provides a shortcut to the information contained in the organizational structure (organized through structural links):

Semantic link Such a link represents an existing semantic relationship between concepts within the domain.

Each `Link` can have the following properties [40]:

Parameters These allow the passing of information between the source node and the target node. Such a parameter gives the instantiation of the target node a reference to one of more instantiations of the classes in the object chunk of the source node instantiation, thus providing access to the source node instantiation's information.

Conditions These restrict the availability of the link for different users, devices or timeframes.

Although HTML only allows one-to-one links, WSDM allows the specification of one-to-one, one-to-many and many-to-one links. These types of links allow to keep semantically or logically related links together. When generating HTML presentations, these links are of course transformed in one-to-one links and the link-types can then be used to determine the type of presentation (e.g., a one-to-many link can be presented as a single menu).

Note that these links link together *concepts*, meaning that a one-to-one *concept* link may cause the creation of several hyperlinks, between for example one *instantiation* of the source node and several *instantiations* of the target node.

5.3 Site Structure Model

The WSDM Site Structure Model contains the concepts needed to map the navigational model onto pages.
A `WebSite` (concept) has one or more `Pages` (concept). This `Page` concept groups `Node` concepts from the navigational model. When these `Node` concepts have links to `Node` concepts within the same page, these links result in links within the page (or no links), when the `Node` concepts have links to `Node` concepts within another page construct, these links will result in links between pages.

More than one site structure may be defined, for different devices, contexts or platforms.

Again, note that a single page *concept* can result in multiple page *instances* in the implementation, resulting from the population of the classes in the related object chunks [40].

5.4 Presentation Modeling Concepts

The WSDM Presentation Modeling Concepts can be divided in two large groups [40]:

Primitive Presentation Concepts This set of low-level concepts allow the description of any layout, but because of their low-level nature, using these concepts is very labour-intensive:

Complex Presentation Concepts This set of high-level concepts (built using the Primitive Presentation Concepts) correspond to well-known GUI concepts and and are easy to use for developers.

The following Sections give an overview of these primitive and complex presentation concepts.

5.4.1 Primitive Presentation Concepts

A hierarchy of the WSDM Primitive Presentation Concepts can be found in Figure 5.4 (taken from [40]).

Two main concept types can be seen in Figure 5.4: `PositioningConcepts` and `FormConcepts`. The latter determines the position of objects on the webpages and the former allows the representation of form elements.

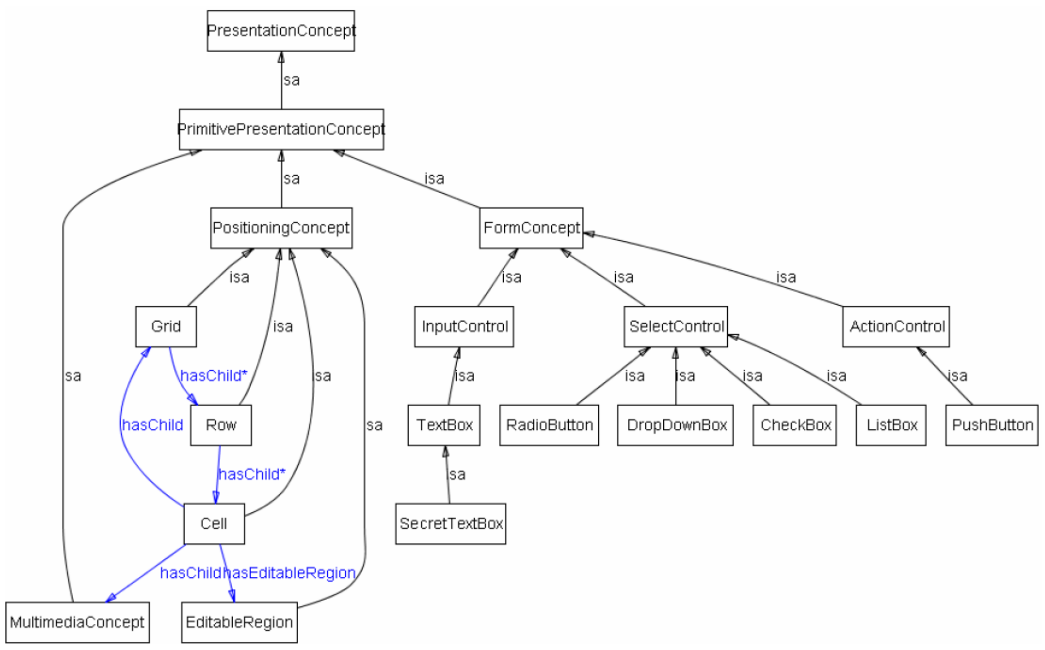


Figure 5.4: WSDM Primitive Presentation Model Concepts.

The positioning of elements happens through a `Grid`, containing `Rows` with `Cells`. These Grid elements can have an absolute or relative height and width, allowing for a complete positioning of their contents. These contents are located within cells, which contain multimedia values (from `ObjectProperties` to `MultimediaConcepts`) or other, nested cells. These cells (and thus, their implemented contents) can be augmented with the navigation links from the navigational model. As mentioned in the previous Sections, multiple instances of a class of a represented Object Chunk can cause, in this case, the repetition of the grid structure for each instance.

`FormConcepts` represent form elements. The WSDM Primitive Presentation Concepts contain `SelectControl` (e.g., selection list), `InputControl` (e.g., textbox) and `ActionControl` (e.g., button) concepts and their sub-concepts. The WSDM Behavior Modeling Concepts allow the attachment of behavior to these `Control` concepts.

5.4.2 Complex Presentation Concepts

Figure 5.5 provides an overview of the Complex Presentation Concepts. These Complex Presentation Concepts are all translated into Primitive Presentation Con-

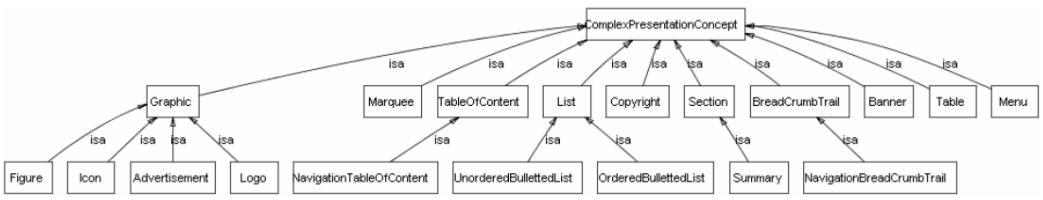


Figure 5.5: WSDM Complex Presentation Model Concepts.

cepts during the WSDM Implementation phase (see Section 3.1.5 on Page 21). Most of the `ComplexPresentationConcept` subtypes are selfexplanatory, so we limit our descriptions to the `BreadCrumbTrail` concept as an example. Such a `BreadCrumbTrail` concept is used to keep track of the path the user followed to reach a certain page.
A `NavigationBreadCrumbTrail` concept adds a link to each item of the path. These items are represented by multimedia items with a list representation.

5.5 Behavior Modeling Concepts

The Behavior Modeling Concepts allow the association of behavior with the presentation model concepts from the previous Section. Figure 5.6 (from [40]) gives an overview of these concepts. The behavior (`Behavior` concept) of a Presen-

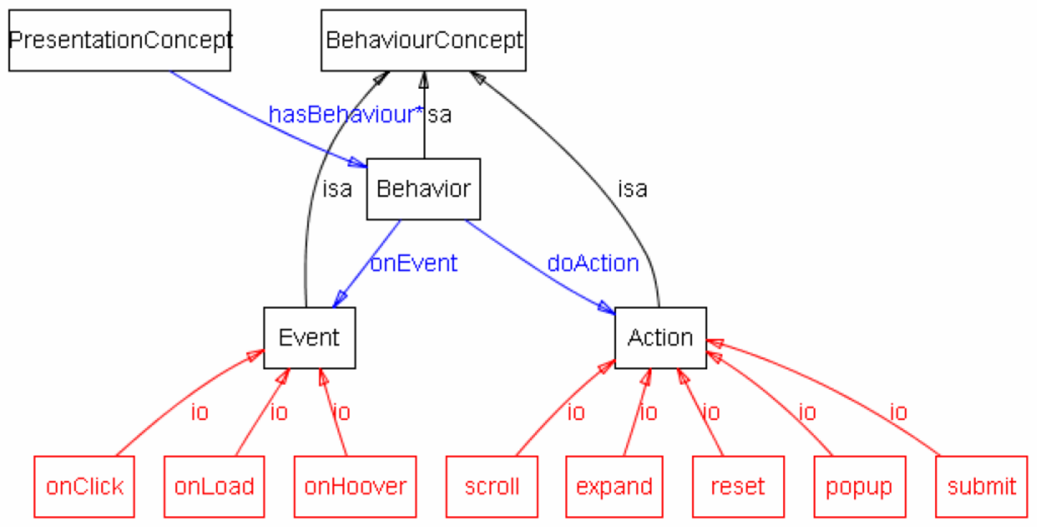


Figure 5.6: WSDM Behavior Model Concepts.

tation Concept expresses the fact that when a specified event (`Event` concept) occurs for that Presentation Concept, the specified action (`Action` concept) will be performed. Not all presentation concepts support all events and actions.

5.6 Style & Template Model

The Style & Template model consists of the following:

Styles These define the font, colour, etc of the page objects through the use of stylesheets;

Templates These define a common layout for several pages through the use of the modeling concepts shown in Figure 5.7.

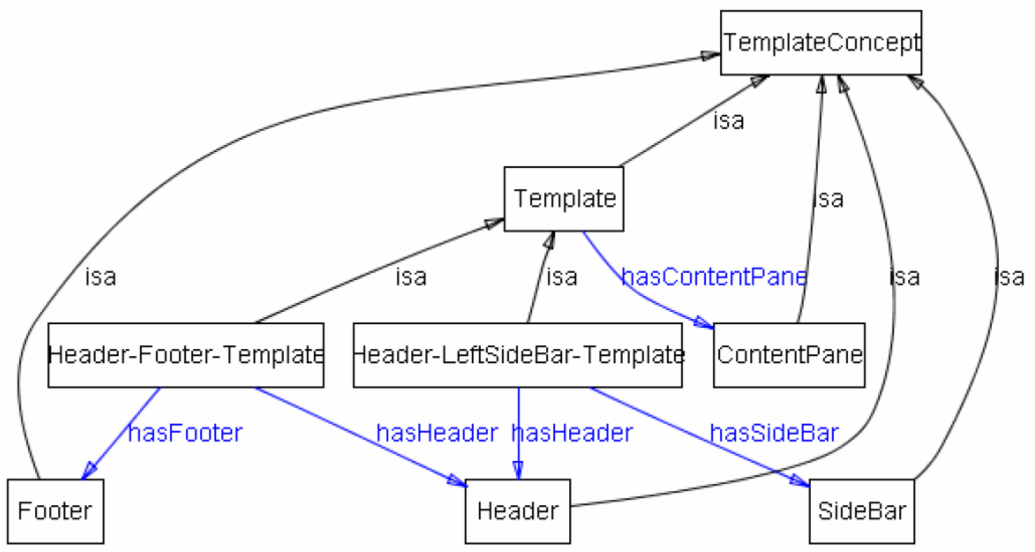


Figure 5.7: WSDM Template Modeling Concepts.

Figure 5.7 shows two examples of predefined templates, namely the `Header-Footer-Template` and the `Header-LeftSideBar-Template`. A template allows the positioning of fixed content (content that is the same for all pages that use the template) and variable content (content that differs from page to page). Fixed content is expressed through presentation concepts (see Section 5.4) attached to a `ContentPane`, `Footer`, etc. Variable content is expressed through the inclusion of the ”EditableRegion” concept which can be ”filled in” during the Page Design.

5.7 Page Model

The layout of the `Page` concepts from the Site Structure Model is addressed by the Page Model. This layout is set using the `TemplateConcept` concepts from the previously described model. These `TemplateConcept` concepts are attached to the `Page` concepts using the `BasedOn` property of the `Page` concept. The editable regions of the Templates are defined using `PresentationConcept` concepts from the Presentation Model.

Part II

Research

Chapter 6

Changes

To be able to correct website errors, we first need to study the cause of these errors. As mentioned in the Introduction (see Section 1.1), the common way to describe website errors is through the concept of ”broken links”. Such a broken link results from the unidirectional and decentralized character of links to resources in the WWW. This means that third parties can create links to resources of a website, without the owner of the resource knowing about this link. A broken link then occurs when the owner of the resource *changes* this resource, without the knowledge of the third party. When the link from the third party then attempts to retrieve the resource, it may fail to do so and ”break”.
Before we go any further, we need to define the terms ”*resource request*” and ”third party link”:

resource request A resource request is the request sent to a website for a certain resource. A prominent example for a resource request is an URL. This type of resource request is then used in the context of a Hypertext link.

third party link A third party link is a link that is made to a resource of the website that is the target of our approach, without the knowledge of the engineer of the website. This type of link can be created by the engineer of another website, or by a user of the website, under the form of a ”bookmark”. A third party link always originates from a link existing on the target website.

Since we use the WSDM Ontology Based Website Design Method (as described in Section 3.1) as a basis for designing our websites, the models of this method capture the entire website. To change a website designed using the WSDM Method means changing the models of the design.

The following Sections describe the changes that can occur on a website and how these changes reflect on the models of the WSDM Method. Section 6.1 describes the possible changes that can be applied to a website. Section 6.2 determines which of these changes are not within the scope of this thesis, and argues why. We then conclude this Chapter with Section 6.3, which lists the changes that are the target of the approach described in this thesis and gives the reasons why this is the case.

6.1 Possible Changes

The following Sections list the changes that can happen for a website and identify the models of the WSDM Method affected by these changes. To be able to do this, we first need to clarify the differences between content and resources:

content Simply put, content is that which is *contained* within a resource. In a webpage, content is text, images, etc. Content is represented as instances of the `MultimediaConcept` concepts of the WSDM Method (See Section 5.1).

resource Resources contain content and are addressable. For a website, a resource is a webpage. Although multimedia sources (e.g.: images) are also addressable, we consider these multimedia sources to be content and thus contained within a resource. In the models of the WSDM Method, a resource corresponds to a `Node` concept of the Navigational Model (see Section 5.2) grouped onto a Page from the Page Model (see Section 5.7).

Now, we are ready to list the changes that can happen for a website:

changing the style and layout This change results in a changed look and feel of the website. Depending on the implementation platform, this look and feel may be embedded in the content of the resources. In the WSDM Method however, the look and feel of a website is captured by the Style & Template Models (see Section 5.6), which are linked to the content by the Page Model (see Section 5.7).

moving resources This change results in a resource being moved from one location to another (e.g.: a webpage is moved from one folder of the webserver to another). In the WSDM Method, these resources are represented by Nodes, grouped onto pages by the Page Model (see Section 5.7).

renaming resources The result of this change is a resource of which the name has changed. In the WSDM Method the Nodes of the Navigational Model (see Section 5.2) are identified by a name property. This name can change, constituting a rename of the associated Page resource.

removing resources A resource has been removed as a result of this change. Because the WSDM Method presumes internal consistency, the removal of a Node from the Navigational Model results in the removal of any references to this node in the Page Model.

removing content The result of this operation is a content element that has been removed. A ”content element” for the WSDM model exists on two levels: the data level and the schema Level. Content on the schema level is present in the WSDM models as a class that is a part of an Object Chunk (see Section 5.1). Content on the data level corresponds to an instance of one of the classes that represent the same content on the schema level. Because of the internal consistency of the WSDM models, the removal of a class from an Object Chunk means the removal of all the instances of that class. Both in the case where a class is removed and in the case when a single instance is removed, the internal consistency requires that all references to these content elements are also removed.
In some cases, the content that has been removed needs to stay removed (e.g. to avoid copyright infringement, to avoid unauthorized access to sensitive information, etc...).
Another special case is the removal of *multimedia sources*. The result of this type of change is that a certain multimedia source, an audio, video or image file has been removed. On the data level, WSDM multimedia content elements are WSDM MultimediaConcept concept instances, but these instances only contain a *reference* to the multimedia source, not the multimedia source itself.

renaming content The result of this change is a content object for which the *name* has changed. The *name* for this type of change means the key that identifies this content object within the content source. Within the WSDM Object Chunks, this key exists on both the data and the schema level. On the schema level, the key is the name of a class contained in an Object Chunk. On the data level, the key is a part of the ”content element” itself, since it is a property of an instance of one of the classes of the Object Chunk.

changing content The result of this change is that one content element of a resource has changed. In the WSDM Method this means that the values for one of the properties of an instantiation of a class that is part of an Object Chunk have changed.

changing the target platform The result of this change is a that a different output format is used to present the website to the user. In the WSDM Method the target platform is specified as a part of the Transformation Pipeline of the Implementation phase (see Section 3.1.5). A change in target platform results in the creation of another Transformation Pipeline, suited for this platform.

incomplete removal of resources The result of this type of change is that a resource has been removed from the website, but references to this resource still exist. This type of removal would cause inconsistencies in the models of the WSDM Method.

changing the domain The result of this kind of change is that the entire website is moved from one domain name to another. Domain names are not captured by the WSDM Method, except in those cases where an absolute url is used to reference the multimedia sources used as values for MultimediaConcept instances' values.

6.2 Unsupported Changes

The changes that are not supported by our approach can be divided in two categories:

Inapplicable Changes These changes have no consequences for the approach presented in this thesis and monitoring these changes would produce no result whatsoever;

Changes Beyond The Scope This type of changes are changes that can be considered as useful to detect, but are beyond the scope of this thesis.

The following Sections define which changes belong to these two categories and provide the reasons why this is the case.

6.2.1 Inapplicable Changes

Inapplicable Changes are changes that can be made to a website, but that are of no consequence for our approach. This means that these changes won't cause ”broken links” or direct the user to unwanted information.

changing the style and layout The style of a website means the colour, font size and location upon a page for the objects of that page. A change in the style of a website does not change the way the pages of the website are addressed nor the content of the website. In the WSDM Method, the style part of the Style & Template Model is presumed to be defined through the use of stylesheets. These stylesheets allow the specification of the style of the page objects independent of the page objects, so a change in style does not effect these page objects. The template part of the Style & Template model defines the location of the content in the different Page concept instances. Each of these Page concepts of the Page Model has an associated Template from the Template Model. These Page concepts also contain the style specifications for the elements of the Page that are not specified by the template.

Since the addressing of resources in the WSDM Method is taken care of by the Navigational Model, and this model is only indirectly (through the Page Model) associated with the Template Model, a change in this Template Model does not affect the addressing of resources in a website created with the WSDM Method.

6.2.2 Changes Beyond The Scope

This type of changes will cause website errors, but the solution of these errors cannot be achieved by using the design models of the WSDM Method.

incomplete removal of resources A badly executed removal of resources can cause a number of errors. The most prominent examples of these are ”broken links” (coming from *within* the website) and ”broken images” (an image file has gone missing). For these kinds of errors to exist in a website created using the WSDM Method, the models of the WSDM Method of this website would have to be in an inconsistent state.

Although it would be useful to fully address this type of change causing the inconsistent state, our error-correction approach only provides partial support for this type of change. Partial, because the ”broken links” caused by model inconsistencies are corrected by our approach to resolve ”broken links” created by third parties as described in Section 6. When it comes to other errors caused by model inconsistencies, we assume that the models of the WSDM Method are kept in a consistent state using a consistency checking method.

changing the domain A change in the domain of a website is enough to invalidate all the third party links made to the website. Unless redirection to the new website has been provided, correcting an error resulting from a domain change is impossible. For websites created using the WSDM Method, a resource request is fulfilled by returning a page based on a number of parameters sent to a generating technology. The location of this technology is determined by the domain of the website. This location is captured by the Transformation Pipeline of the WSDM Method. If the original domain is still under the control of the website engineer, the problems caused by a changed domain can be solved by redirecting the resource request sent to the original domain to the new domain, with the location of the generating technology changed to reflect the new location. This type of redirection can be easily achieved through any standard redirection technique available. The usage of this kind of solution does not directly relate to the advantages of using an Ontology Based Website Design Method and because of this, will not be treated in this thesis.

6.3 Supported Changes

The changes that are supported by our approach can be divided in two categories:

Changes Captured By WSDM These changes are captured by WSDM, meaning that websites designed using the WSDM Method do not suffer the consequences of these changes, thereby limiting the number of problems that need to be solved;

Changes Captured By Our Approach This type of changes are the changes that are captured by the error-correction approach proposed in this thesis. These changes cause the occurrence of ”broken links” or may cause the user of a third-party link to be guided to unwanted information.

The following Sections define which changes belong to these two categories and provide the reasons why this is so.

6.3.1 Changes Captured By WSDM

This Section contains the changes that would cause website errors for websites implemented using ordinary architectures, but that are prevented to do so by using the WSDM Method as the design method for the website.

moving resources Normally, when a resource (e.g. a webpage) is moved within a webpage (e.g. moved to a different folder), the previous location of this resource can no longer be used as a valid resource request. A third-party link that existed prior to this move and that contained the now invalid resource request has now become a broken link. This kind of error does not occur for websites designed using WSDM Method. Moving a resource of a website that was designed by the WSDM Method simply moves this resource onto another Page. But, in WSDM, resources are addressed directly and not through these Pages, so the way of addressing this moved resource will not change.

changing the target platform A change in the target platform normally results in a change in the composition of the resource requests of a website. The target platform is usually expressed through the usage of extensions (e.g.: *.html for Hypertext pages and *.wml for WAP pages) for page files. Since these extensions turn up in the resource requests, a change in target platform means invalidating the resource requests for existing links. In websites created using the WSDM Method, extensions not a part of the resource request. The Transformation Pipeline of the WSDM Method captures the target platform and the different models of the WSDM Method are thus not affected by this type of change. This includes the Navigational Model, the model specifying the structure of the resource requests for websites created by the WSDM Method.

6.3.2 Changes Captured By Our Approach

The changes captured by our approach are the changes that can cause errors when considering third party links to webpages designed with the WSDM Method.

renaming resources Renaming a resource affects the resource request for that resource. Whether the resource is contained in a file on the server or generated by the server, renaming the file or the identifiers sent to the generating technology invalidates the existing links using the old resource requests. The different elements that make up a resource request in the WSDM Method are identified through names, a change in these names means a change in the resource request.

removing resources Removing a resource means that the resource request will request a resource that is no longer there. In the context of a website designed using the WSDM Method, this means that the resource request refers to a Node that is no longer part of the Navigational Model, thus invalidating the resource request associated with an existing third-party link.

renaming content Since a resource request contains references to the identifiers used to retrieve content, a change in these identifiers invalidates the resource requests associated with existing third party links.

removing content Removing a content element means that the requested resource no longer contains the content element requested. In the WSDM Method the request for the content element is contained within the resource request. When this content element has been removed, this means that the resource request refers to a content element that is no longer part of the Navigational Model, thus invalidating the resource request associated with an existing

third-party link.
When the removed content element was removed due to its sensitive nature (e.g.: copyrighted or insecure content), the resource request associated with the existing third-party link has not only become invalid, but it is also no longer *allowed* to retrieve the content it targets.
When it comes to removed multimedia sources, there are two perspectives to consider:

The internal perspective From the viewpoint of the website, a multimedia source is a content element, and it should be treated as a content element vulnerable to the ”changing content” and ”removing content” changes.

The external perspective From the viewpoint of the a third-party, a multimedia source can be considered as a resource. The resource request for the multimedia source can be determined by the third-party and a link to this resource can be created. However, since such a direct link to a multimedia source is prohibited by both the implementation (the website engineer never intended for a multimedia source embedded within a resource to be directly linked) and the WSDM Method (links directly to multimedia sources are unspecified in the WSDM Ontology), this type of link should not be considered when considering changes to resources.

So when considering the content-related changes, we need to include multimedia sources as one of the types of content affected.

changing content Changing the content of a resource usually does not invalidate the resource request. However, the user of the third-party link may not get the content that was *expected*. When there is a possibility for a certain content object of a resource to change, it is up to the website engineer to specify whether a link to this type of content allows the content to change or not.

Chapter 7

Supporting Changes

This Chapter describes our error-correcting approach and specifies in which way the changes determined in Chapter 6 are supported by this approach.

7.1 Goals

To be able to specify the goals of our error-correction approach properly, we first need to define the meaning of a ”*recovery*”:

recovery The recovery of a content object or a resource means that we are able to *recover* from the results of the changes performed on that content object or resource. A clear example of this is restoring a resource that has been removed to be able to fulfill a request for that resource.

The goals of our error-correction approach are the following:

- To provide recovery from the specified changes whenever applicable.
- To do so taking into account the intention of the website engineer.

In the following Sections we will specify how these goals are reached. Section 7.2 will first provide an overview of the solutions proposed by this thesis and Section 7.3 will provide additional details.

7.2 Solution Overview

This Section starts by describing the architecture of our approach in Section 7.2.1, followed by Section 7.2.2 specifying the different components that are a part of our approach and finishing up by Section 7.2.3, which describes the general process of our approach.

7.2.1 Architecture

Figure 7.1 gives an overview of the architecture of our solution. As shown in

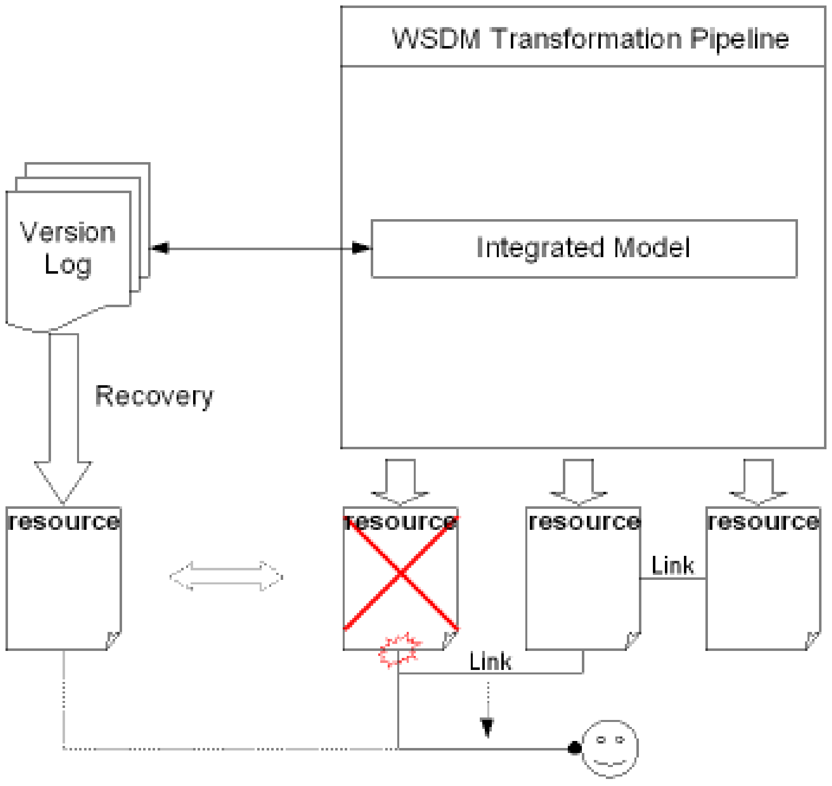


Figure 7.1: An overview of the error-correction Architecture.

Figure 7.1, our approach provides a way to recover resources that have changed or have been removed. Recovery of these resources is needed to fulfill resource requests associated with **links** made by third party users. To achieve recovery, our approach creates a **Version Log** for an intermediate model of the **Transformation Pipeline**. We will refer to this intermediate model as the **Integrated Model**. As described in Section 3.1.5, the Transformation Pipeline implements websites designed with the WSDM Method. The Version Log that we create for the Integrated Model of the Transformation Pipeline records all the changes that are made to the Integrated Model during its lifetime.

The next Section specifies the components of our architecture.

7.2.2 Components

The different components of our approach as shown in Figure 7.1 are described in this Section. Section 7.2.2.1 describes the way links among resources are specified in the WSDM Method and how these aid our approach. Section 7.2.2.2 situates the usage of the WSDM Transformation Pipeline within our approach. The next Section, Section 7.2.2.3 describes the advantages of using the Integrated Model in our approach. Section 7.2.2.4 shows how the Version Log component ties into our approach. And finally, Section 7.2.2.5 describes the usage of the Change Definition Language within our solution.

7.2.2.1 WSDM Links

Figure 7.2 illustrates the way links between resources are created according to the WSDM Method (as described in Section 3.1.4.1). The WSDM Method rep-

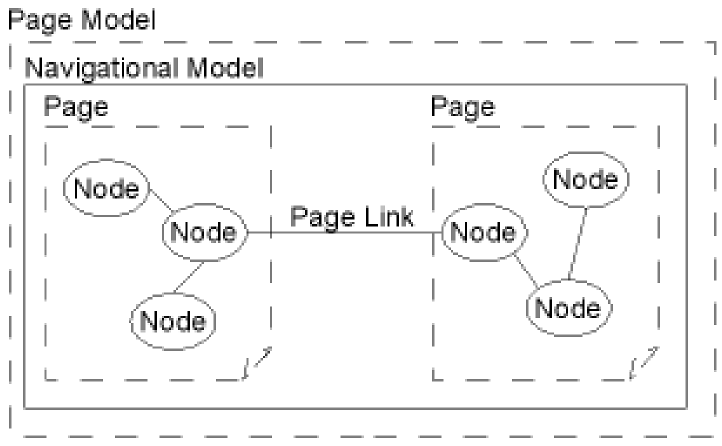


Figure 7.2: The creation of pages in the WSDM Method.

resents the content of a resource on the conceptual level by the Nodes from the Navigational Model. Such a Node from the Navigational Model references an Object Chunk from the Information & Functional Modeling phase of the WSDM Method. An Object Chunk in turn captures the information needed to fulfill the functional and informational requirements of a task from the WSDM Task Modeling phase.

The links between these Nodes are the origin of the links between pages. It is only when the Page Model is overlayed onto the Navigational Model that it becomes clear which Nodes are grouped onto pages. When a links exists between two Nodes on different pages, this link is the actual link between these pages, the WSDM Method does not use different constructs to express links between Nodes

grouped on the same or different pages.
These links between pages determine the resource requests possible for the website and thus also determine the third party links possible.
Since each link refers to one or more Nodes and Nodes represent content on the *conceptual* level, each link needs one or more references to content on the *implementation* level. This type of references are called the Parameters of the link, and they reference the instances of the classes associated with the Object Chunks (Referenced by the Nodes). These instances contain the content as values (e.g.: strings) for their properties.

7.2.2.2 WSDM Transformation Pipeline

The WSDM Website Design Method uses a number of models to capture the entire structure of a website. This structure is divided over the following models:

- The Object Chunks, (see Section 5.1).
- The Navigational Model, (see Section 5.2);
- The Site Structure Model, (see Section 5.3);
- The Page Model, (see Section 5.7).

These models are then used as input during the Implementation phase of the WSDM Website Design Method (see Section 3.1.5). The process that *transforms* these models into a website is called the WSDM Transformation Pipeline. Figure 7.3 focuses on the Transformation Pipeline, as a part of the architecture of our approach from Figure 7.1. The Transformation Pipeline consist of a number of phases, including the Model Integration phase. This phase is needed to be able to map the models onto a number of pages for an implementation platform. The information necessary to construct a single page, is spread out over the models listed above, the Model Integration phase integrates all these models in one large model, the Integrated Model (IM), which is the subject of the next Section.

7.2.2.3 Integrated Model

The IM captures the structure of the entire website that is the *target* of our approach. It consists of the different models of the WSDM Method merged together. The merging of these models is simply achieved by taking the OWL representations of these models and collecting them in a single OWL file. Since all models use the common OWL WSDM Ontology (see Section 5), this merge is easily achieved. The reason for selecting the IM over any of the other models of the WSDM Method is the following:

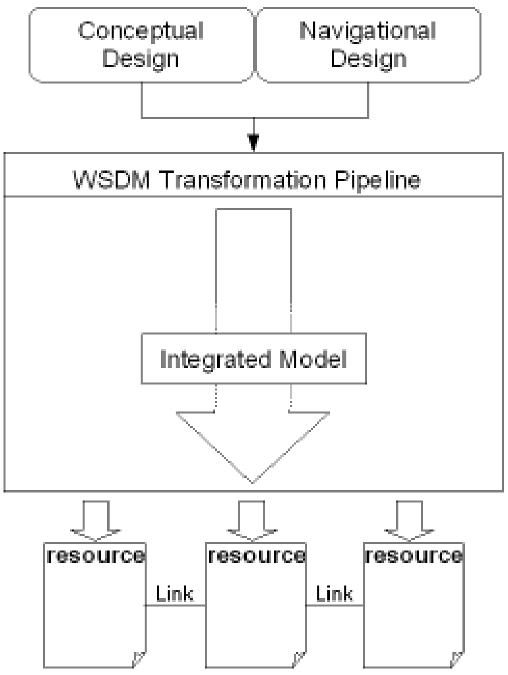


Figure 7.3: The implementation of pages in the WSDM Method.

To increase the separation of concerns, the information gathered by the WSDM Method is spread out over several models. The IM groups these models together, meaning that we only need to keep a Version Log for a single model ontology, as opposed to many. By maintaining a Version Log for only a single ontology, the need to synchronise the chronological order of the Version Logs for several model ontologies disappears.
Because the IM captures the entire website for that implementation of the website, a record of this model would provide a good basis for recovery. The component that provides this record, the Version Log, is discussed in the next Section.

7.2.2.4 Version Log

A Version Log is a crucial part of the WISE Ontology Evolution Approach, as described in Section 4.3. Although we do not use the WISE Ontology Evolution Approach in its entirety, we do make use of the technologies developed for the approach. These technologies are the Version Log and the Change Definition Language. This Section discusses the Version Log and the next Section describes the Change Definition Language.

The Version Log is a log that keeps track of the different versions an ontology concept passes through during its lifetime. The concepts used in the Version Log are defined by means of an ontology, called the Version Ontology (see Section 4.3.8). This Version Ontology gives the Version Log the capability to maintain a list of the past and current versions for each concept of the ontology, besides other properties relating to these versions. One of these properties is a *version number* assigned to each version, this version number adds a temporal order to the versions, achieved through the numerical order of the version numbers.

By keeping track of all the concepts of the IM, the Version Log provides a backup of the different versions of the IM. Each of these versions is recorded as a result of a change being made to the IM. During the different iterations of the WSDM Method, the IM will go through many intermediate changes before the Implementation phase is reached. These intermediate changes can cause the IM to (temporarily) be in an inconsistent state (e.g.: a concept is removed from the IM while the references to this concept have not yet been removed). When the IM reaches the Implementation phase, it will off course need to be in a consistent state. The IM that is used as input for the Implementation phase represents a finished product from the viewpoint of the website engineer.
Since the Version Log records all these changes as additional versions, we will need to differentiate between the intermediate versions and the *”website”* versions.

The Version Log provides *versioning* for the IM, to be able to use this source of information, we need an effective way to query the Version Log. The Change Definition Language of the WISE Ontology Evolution Approach described in the next Section, provides this functionality.

7.2.2.5 Change Definition Language

The chronological order within the Version Log, as described by the previous Section, provides the possibility to check properties of past versions of ontology concepts by means of conditions. In the WISE Ontology Evolution Approach these conditions are used to define the changes that should be monitored by the Approach and specified using the Change Definition Language.
In our approach however, the main application of the Change Definition Language is the usage of these conditions to determine which version of the IM should be *restored* from the backups provided by the Version Log. This restoration is needed to provide recovery for an erroneous resource request. The information provided by this resource request in combination with the conditions of the Change Definition Language is sufficient to determine the correct version number.

7.2.3 Process

In this Section we will describe the *general* process used by our approach. Because of the general nature of this description, we will not yet distinguish between the different changes that may have caused the error. Section 7.3.2 contains the detailed process, including the different approaches for these changes.

The general process is as follows:

1. A number of changes have been performed on the IM of a website that uses the WSDM Method. These changes have been recorded in the Version Log, including the state of the IM prior to these changes. If, for example, one of the changes was the removal of the concepts from the IM that represent the ”join” page of a fanclub and the links to that page, the Version Log will record this ”removal change” in the Version Log for these concepts, while also maintaining the ”creation change” and the other modification changes that have happened to these concepts prior to their removal.

2. The changed IM is used as input for the Implementation phase of the WSDM Method. At this point in time, we mark the last version recorded in the Version Log to be a *”website version”*, which means that this version represents the website that is then implemented and presented to the users.

3. The WSDM generation pipeline has now implemented the latest version of the website. In this version, the example ”join” page and the links to this page have been removed. However, without our knowledge, a third party had created a link to this ”join page” prior to its removal.
   The third party then uses this link and sends the resource request associated with the link to our website.

4. Because the resource (the ”join” page) requested is no longer available, the request causes an error. This error triggers the error-correction mechanism of our approach.

5. The erroneous resource request corresponds with a WSDM link to the ”join” page that has been removed from the IM since the previous implementation of the website. We now need to determine the version number of the version in the Version Log that represents this previous implementation. We do this by transforming the erroneous resource request into a query in the Change Definition Language that retrieves the WSDM link for this resource request prior to its removal and the version number associated with this link.

6. We now have the version number of the most recent website version in the Version Log that still contained the WSDM link associated with the erroneous resource request for the ”join” page. Because of the internal consistency of the IM, the version that captures a link to the ”join” page, must also capture the ”join” page itself. By restoring the IM to the found website version that was captured by the Version Log, we can roll back the website design to the previous version.

7. We can now use this restored IM to create a temporary resource (the ”join” page) to return as a response for the resource request that originally caused the error. Thereby effectively correcting the error caused by the resource request.

7.3 Detailed Solution

This Section provides more details for the Solution presented in Section 7.2. Section 7.3.1 describes a number of extensions to the WSDM and Version Ontologies needed to support our approach. Section 7.3.2 provides a more detailed view of the process of our approach that was described in Section 7.2.3.

7.3.1 Component Extensions

The WSDM Ontology of the WSDM Method and the Version Ontology used by the Version Log need a few extensions to accommodate the error-correcting approach presented in this thesis.
These extensions are grouped around four different subjects:

1. Links: an extension is needed to indicate the temporal context of a link;

2. Deletion: an extension is needed to allow the website engineer to specify whether a deleted resource may be recovered;

3. Website Versioning: we need an extension to be able to indicate which versions in the Version Log are *website* versions;

4. Files to Links: an extension is needed to map the files of static websites to WSDM resource requests;

5. Content Versioning: special treatment is needed to provide versioning for certain types of content.

The following Sections each describe one of these extensions.

7.3.1.1 Links Extension

In this Section we present an extension of the WSDM ontology to be able to explicitly include the *temporal context* of the link. This temporal context addresses the issues brought up by the ”changing content” change in Section 6.3.2. The website engineer needs to be able to specify whether a link to a resource allows the contents of that resource to change. This specification is called the temporal context of the link. There are two kinds of temporal context possible: the ”snapshot” type and the ”live-time” type. Figure 7.4 illustrates the differences between these contexts. Figure 7.4 shows two links to the Event Organizer Page concept with

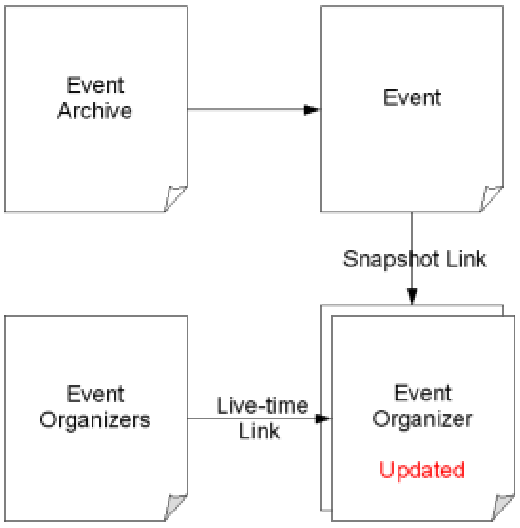


Figure 7.4: Types of temporal contexts for a concert example.

different types of temporal context. The link coming from the Event Organizers Page expects the *most recent* information about a particular event organizer to be displayed on the Event Organizer Page, so the website engineer has specified this link to have a **live-time** temporal context. The link coming from the Event Page that is a part of an Event Archive expects the information about the organizer to be relevant to the event described on the Event Page. We can assume that this was the case at the point in time when the Event Page was created, so upon creation of the link the website engineer specifies this link to have a **snapshot** temporal content.

The example illustrates that there may be cases in which one temporal context allows the content of a resource to change, while another temporal context forbids the content of that same resource to change. To be able to solve this problem, we saw the need to extend the WSDM Link concept.

This extension should provide two things:

- A way to record type of temporal context a link has. We repeat the two types possible:

  Snapshot type A link with this type of temporal context does not allow the content of the resource it links to, to change. A snapshot link provides an *image* of the resource at the time the snapshot was taken. This point in time occurs at the creation of the link.

  Live-Time type A link with this type of temporal context allows the content of the resource it links to, to change. Just like with a regular Hypertext link, a live-time link always points to the most recent version of the resource available.

- A way to record at which point in time the ”snapshot was taken”, for snapshot links. We can take a version number from the Version Log to specify this point in time. To be more precise, the version we take the version number from, is the website version number of the IM that had the snapshot link newly added.

Figure 7.5 describes the `Link` concept from the WSDM Ontology and its properties. This `Link` concept is part of the Navigational Model as described by Figure 5.3 in Section 5.2. The extension of the `Link` concept of the WSDM Ontology consists of adding the `hasContextType` and `hasContextVersion` `DataTypeProperties`.
The `hasContextType` property is a property obligated for each `Link` instance and can take as values the strings ”`live-time`” and ”`snapshot`” indicating the type of temporal context specified by the ontology engineer.
The `hasContextVersion` property is only valid when the context type is ”`snapshot`” and, additionally, the ontology engineer is not allowed to set the value of this property manually. This value of the `hasContextVersion` property is the number of the IM version captured by the Version Log that contains the Link as a newly added concept. The setting of this value is done by our approach, during the WSDM Implementation phase.

Note that this extension constitutes a modification of the resource requests for websites created by the WSDM Method, effectively tying in the WSDM resource requests with the Version Log.

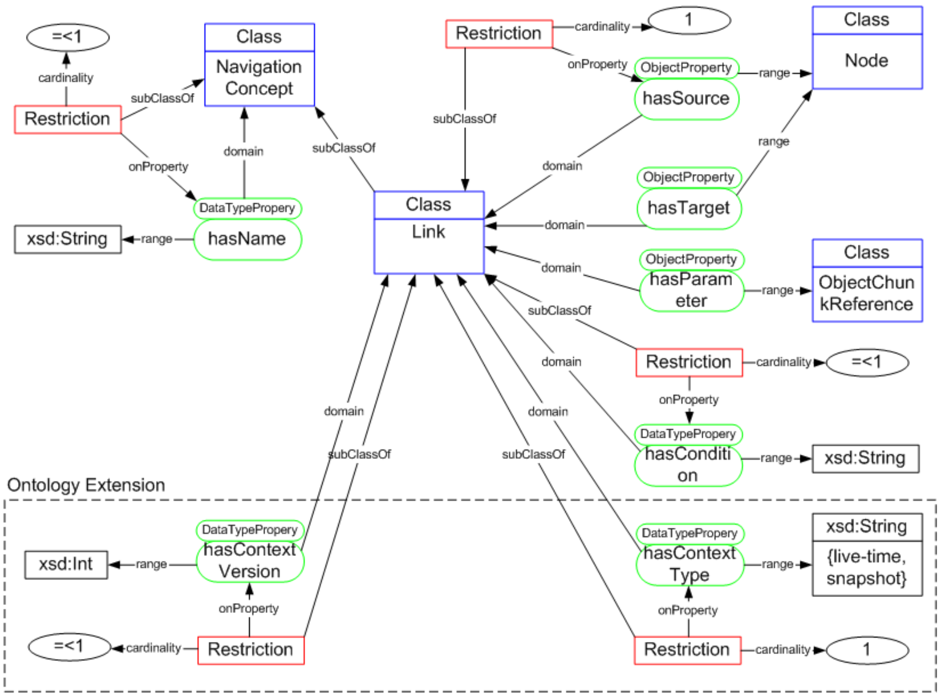


Figure 7.5: The `Link` class and its properties, including the extensions.

7.3.1.2 Deletion Extension

As explained in the specification of the ”removing content” change in Section 6.3.2, sometimes, resources are not allowed to be recovered by our approach, once they have been deleted. Because of this, the Version Ontology used by the Version Log needs to be extended to allow the specification of a kind of ”strong” delete.

Figure 7.6 gives an overview of the Version concept that is part of the Version Ontology as described by Figure 4.9 in Section 4.3.8. The extension of the Version Ontology consists of adding the ”`deleted`” option to the list of valid values for the `hasState` property. Normally, when a concept is removed from an ontology that is being monitored by a Version Log, a new version is added to the Version Log to reflect this removal. This new version for the removed concept receives as value for its `hasState` property the ”`retired`” string. This value may be changed by the website engineer into the ”`deleted`” string to specify that this concept may not be recovered by our approach. If our error-correction approach is invoked by a resource request for a deleted resource that may not be recovered, our error-correction approach will halt the recovery process and return an error page for the resource request.

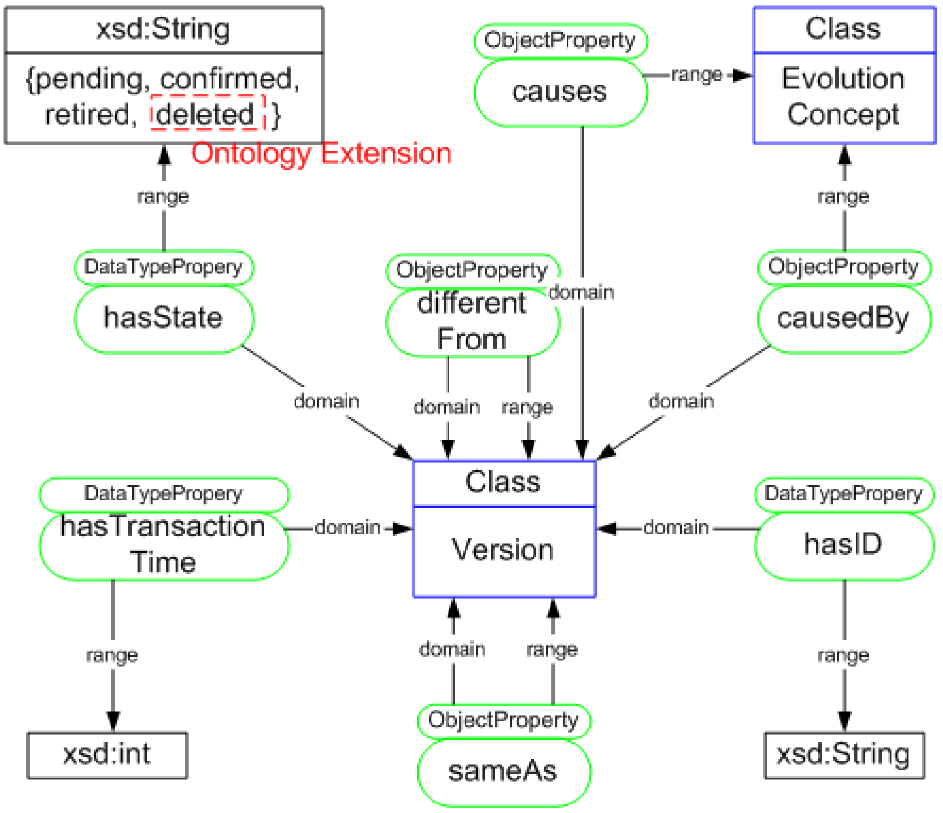


Figure 7.6: The `Version` concept and its properties.

7.3.1.3 Website Versioning Extension

As mentioned in Section 7.2.2.4, we need to be able to distinguish between versions in the Version Log that represent the implementable (or *website*) versions and versions that represent intermediate changes made to the IM. The reason for this is that these intermediate changes may bring the IM (and thus the version in the Version Log capturing the IM at that point in time) into a temporary inconsistent state (e.g.: a concept has been removed while references to this concept remain). And, such an inconsistent IM would not be suitable as input for the Implementation phase of the WSDM Method. Not only do the so-called *website versions* capture a consistent version of the IM, they also represent a finished product from the viewpoint of the website engineer.
Because of these properties, the website versions are the only versions that should be considered to be candidates for recovery.

To store the current version number of the website, we extend the `WebSite`

concept used in the Site Structure Model (see Section 5.3) with an additional property. Figure 7.7 gives an overview of the `WebSite` concept and the extension made.
The extension here is under the form of the `hasVersion` property. The value

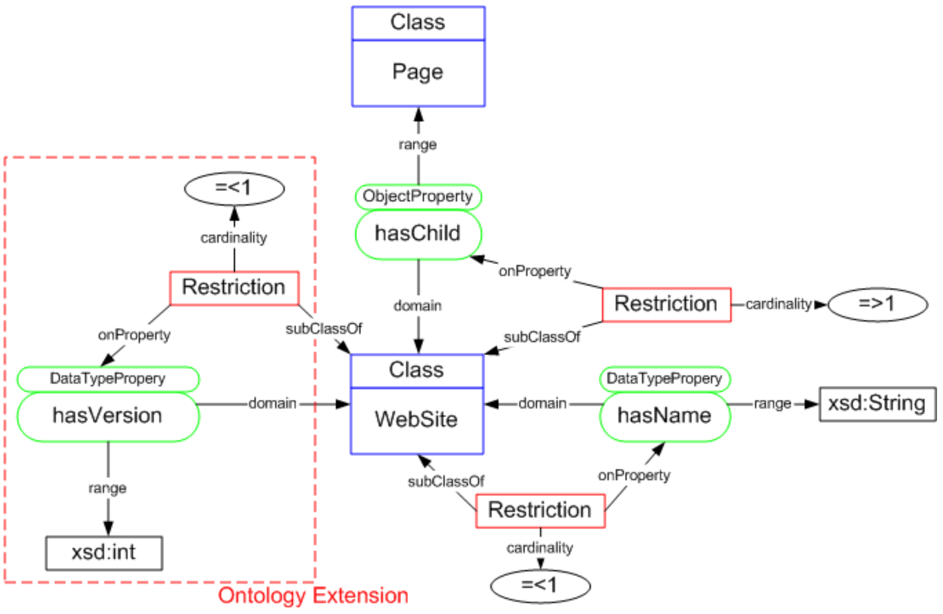


Figure 7.7: The `WebSite` concept and its properties.

that is assigned to this property is the version number of the version that captures the ”add new/modify existing hasVersion property” change. This change occurs right before the IM is presented to the Transformation Pipeline, which makes sure that the version number that is assigned to the `hasVersion` is the last version captured prior to the implementation of the website.
Setting this property in the IM happens automatically by our approach and the website engineer is not allowed to set the value of the `hasVersion` property manually.

7.3.1.4 Files to Links Extension

A website created by using the WSDM Method has been implemented as either a static or a dynamic website. A static website consist of a number of fixed pages in a certain format (e.g. *.html files) depending on the target platform of the implementation. A dynamic website generates the pages on the request of the user, based upon the WSDM models. In the static system, a resource request retrieves a file, while in the dynamic system, a resource request builds the resource and returns it. In fact, the files of the static system correspond to a one-time generation of the resources as if they had been generated in response to dynamic resource requests.

To be able to apply our error-correction approach to static websites created by the WSDM Method, we need to extend the WSDM Ontology to maintain a mapping between the static resource requests (requests for files) and the corresponding dynamic resource requests (requests for concepts of the IM) for the website.

This extension is also added to the `Link` concept of the WSDM Ontology. Since a `Link` concept captures the information of a dynamic resource request, this is the most obvious place to store the static resource request (a filename).

Determining which filename corresponds with which link can be done manually by the ontology engineer or automatically by the WSDM Transformation Pipeline. Figure 7.8 shows the extension from this Section added to the `Link` concept. Note that the extensions from Section 7.3.1.1 have been omitted in Figure 7.8

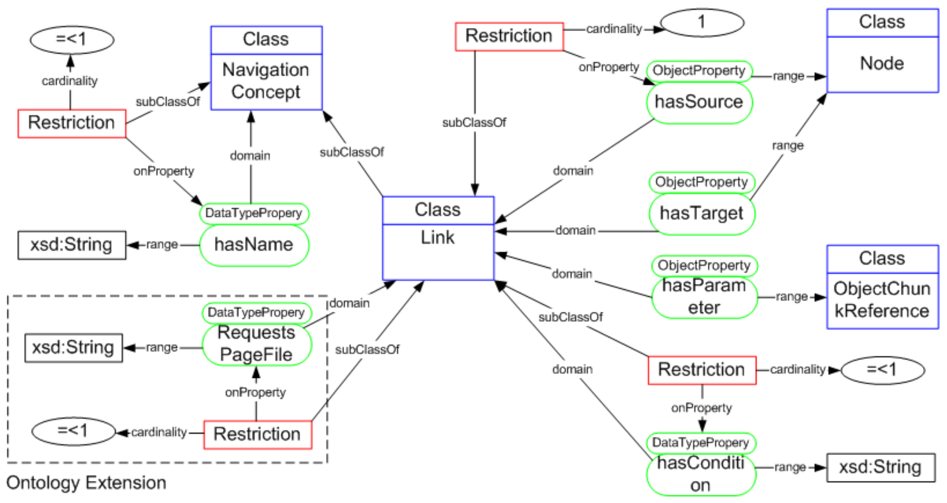


Figure 7.8: The `Link` class and its properties, with only the extension from this Section.

to provide clarity. The extension in this case is achieved through the optional `requestsPageFile` property which contains a filename string representing the static resource request corresponding to the dynamic resource request captured by the `Link`.

Also note that the `requestsPageFile` property should only be added to the `Link` concepts that represent resource requests. The links that exist between Nodes *within* a single Page do not represent resource requests. As has been visu-

alized by Figure 7.2 on Page 85, only the links between Nodes on different Pages represent resource requests.
Since multiple resource requests can exist for a single resource (meaning multiple links to a single Page can exist), and a single resource results in a single file, multiple Link concepts will contain the same filename. When our approach then needs to retrieve the resource request from this filename, we will simply accept the first resource request retrieved since each of the retrieved resource requests for the resource are equal.

7.3.1.5 Content Versioning Practice

For websites created using the WSDM Method, the content (pieces of text, images, audio, etc..) of the website is kept separate from the models describing the website structure up to the Implementation phase, when the content is mapped to the IM. The WSDM Method supports various sources of content that can be mapped to the IM. These content sources can be divided into two groups:

DataBase Sources Versioning for DB systems is available under the form of *Temporal Databases*[1]. This type of database provides the ability to roll back data by recording the ”Transaction Time” (point in time when the data was entered or superseded) for the data within the database.

File Sources When files are used as a data source, our approach requires a form of versioning for these files. The most obvious form of versioning available is by using a Version Log for an OWL-file based datasource.

In this thesis we will assume that OWL files are used as a content source. These files are then integrated into the IM and versioning is provided through the Version Log for the IM.
However, there are limitations to the types of content that can be captured in OWL. Multimedia data is represented in OWL by an URL referring to the location of the *file* containing the multimedia data. The versioning thus only provides versions for the location of the data, not the data itself.

As mentioned in Section 6.3.2 with the description of the ”removing multimedia sources” change, we need to provide versioning for the content, even for the multimedia data itself. We achieve this versioning by simply creating local copies of these files and updating the references to these *versioned* files in the Version

[1]Temporal Database on Wikipedia, see `http://en.wikipedia.org/wiki/Temporal_database`

Log. So whenever a new multimedia file has been added to the website, our approach creates a single backup copy for this file, and references this copy as the multimedia source in the Version Log.

7.3.2 Detailed Process

In this Section, we will describe how the different cases that may be a part of the recovery process are treated in our approach. We will start by describing how our recovery process responds to the different changes that were described in Section 6.3.2 in Section 7.3.2.1, followed by Section 7.3.2.2 that contains the description of several special cases that are caused by the extensions needed by our approach (see Section 7.3.1).

7.3.2.1 Handling Changes

This Section describes the algorithms for handling the changes from Section 6.3.2. Since all these different changes are all part of one larger process to correct erroneous resource requests, we first describe this process. To do so, we first distinguish between the two different types of resource request a website created by the WSDM method can receive, namely:

1. Resource requests with a live-time type of temporal context;
2. Resource requests with a snapshot type of temporal context.

Figure 7.9 describes the full process for live-time type of resource requests.
The different processes from Figure 7.9 are the following:

Check Existence We check whether the resource request can be fulfilled. We do this by executing the query specified in Listing 7.1. If the resource request can be fulfilled, we simply do so, otherwise, our error-correction mechanism is triggered ad we continue with the next step;

Retrieve Website Versions We get the website version numbers recorded in the Version Log. We get these version numbers through the query from Listing 7.10;

Find Link For each of version numbers retrieved from the previous step, we try to find the link corresponding to the resource request. The query from Listing 7.2 is used to find this link;

Find Renamed We now need to determine whether a renaming change has occurred. The query from Listing 7.3 does this for the link that we have retrieved in the previous step. If the renaming change has been detected, we

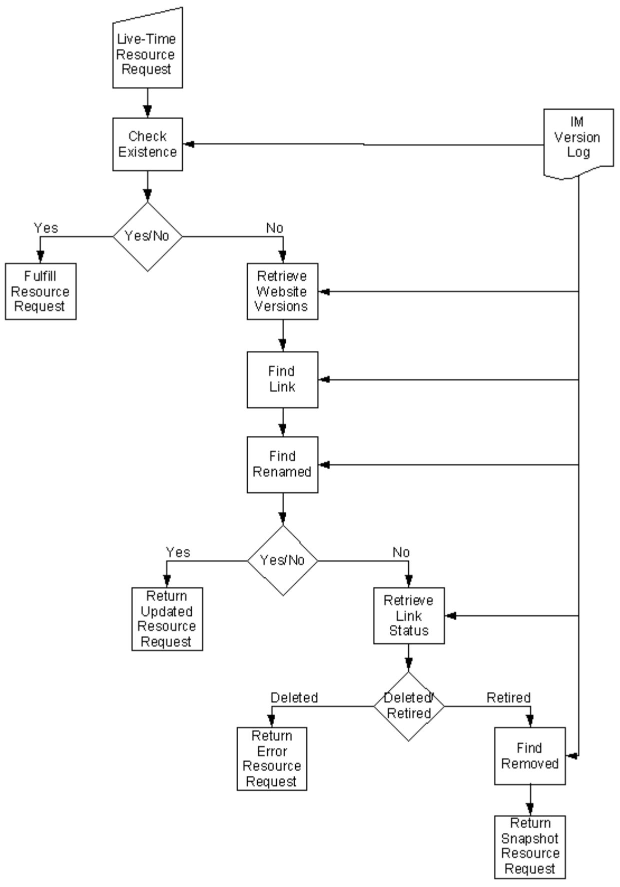


Figure 7.9: The detailed process for live-time resource requests.

redirect the erroneous resource request to an updated resource request that reflects the consequences of the renaming change. If the renaming change has not been detected, a removing change has been detected and we continue with the next step;

Retrieve Link Status We retrieve the status of the version capturing the link we have retrieved in the previous steps. If this status is ”deleted”, we redirect the erroneous resource request to a request for an error page, since the removed resource may not be recovered. If the status is ”retired”, we can continue with the next step. The retrieve of the status happens through the query from Listing 7.8;

Find Removed We have now detected a removing change, and the removed resource may be recovered. We retrieve the version number of the version from the Version log to recover, and the resource request associated with the link concept of that same version. Retrieving this information is done through the query of Listing 7.4;

Return Snapshot Resource Request Since we now have the version number and the resource request from the version with that retrieved version number, we can create a snapshot resource request. This snapshot resource request will recover the version of the IM that is needed to fulfill the resource request found in the previous step.

Figure 7.10 describes the process for the snapshot type of resource requests. The different processes found in Figure 7.10 are:

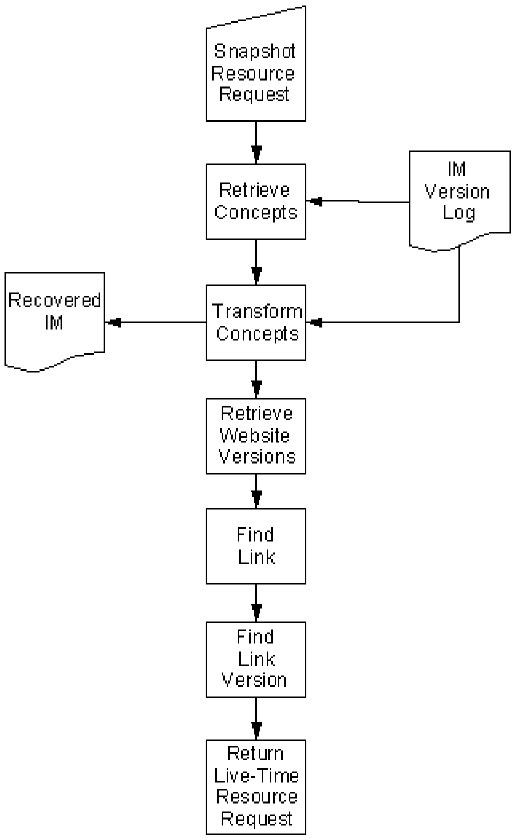


Figure 7.10: The detailed process for snapshot resource requests.

Retrieve Concepts We collect the concepts of the Version Log that have as version number the contextversion number requested by the snapshot resource request. This is done through the query in Listing 7.5;

Transform Concepts We transform the concepts we have retrieved in the previous step back into an the IM that was captured by that version of the Version Log;

Retrieve Website Versions We get the website version numbers recorded in the Version Log. We get these version numbers through the query from Listing 7.10;

Find Link For each of version numbers retrieved from the previous step, we try to find the link corresponding to the resource request. The query from Listing 7.2 is used to find this link;

Find Link Version We now get the resource request associated with the link we found in the previous step, from the version with as version number the contextversion number from the snapshot resource request. The query to retrieve the parameters for this resource request is listed in Listing 7.6;

Return Live-Time Resource Request We now have retrieved the resource request and the IM for the contextversion number of the snapshot resource request. The resource request can now be fulfillled as a normal live-time resource request by the temporarily implementation of the IM.

The following Sections now each describe the algorithms for each change type separately.

7.3.2.1.1 Handling The ”renaming resources/content” Changes

As mentioned in Section 6.3.2, the different elements that make up a resource request in the WSDM Method are identified through names, a change in these names means a change in the resource request used to retrieve this resource. An example of this type of change is a live-time link that originally uses the following identifiers to retrieve a webpage resource that describes a member of a fanclub:

DescribeMember This is the name of the node from the Navigational Model that is requested by the link;

Member This is the name of the class that identifies the content on the schema level;

John This is the key of the instance of the Member class that identifies the content on the data level.

The name of the class used to identify the data on the schema level has now *changed* from ”Member” tot ”FanclubMember”. This renaming change has instantaneously changed all the resource requests for the fanclub members’ informational content.

Note that we have grouped together the renaming changes for content identifiers as well as for resource identifiers, since the treatment of both changes is equal.

We make the following assumptions that determine the current state of the environment:

- A complete Version Log exists and the IM is internally consistent;
- The resource request is of the temporal context type ”live-time”, since this type of change does not apply to resource requests with a ”snapshot” temporal context;

We start by giving a step-by-step description of the algorithm used, followed by a more detailed description of the main queries used.

Algorithm

1. First, we need to check whether the resource request that we have received will cause an error or not. We do this by executing the first of the three queries described in the next Paragraph. If this query returns a result, this means that our website will be able to fulfill the resource request. If this is not the case, we continue with the next step of our algorithm.

2. Next, we determine the type of change causing the resource request to be erroneous. There are two types of change possible, the renaming change or the removing change. The removing change will be treated in Section 7.3.2.1.2, while the renaming change will the treated here.
   To determine this type of change, we use the second and third queries described in the next Paragraph.

3. One of two cases occurs as a result of the execution of these queries. When we receive output from the third query, it is the presence of this output that confirms that the renaming change has taken place. Similarly, when we do not receive any output, this means that the removed content change has taken place, in this case, we will then need to continue with the solution presented in Section 7.3.2.1.2.

4. If there is output provided by the second query, then this output consists of the changed identifiers that were the result of the ”renaming” change. These identifiers can then be used to create an up-to-date resource request. For our example the identifiers received would be ”DescribeMember”, ”FanclubMember” and ”John”. From these identifiers we can construct a new, correct resource request.

5. Finally, we can now redirect the erroneous resource request to the created up-to-date resource request, hereby correcting the error.

Description of the Version Log Query

- We start with the query from Listing 7.1. This query determines if the current version of the Version Log (and thus the current version of the IM) contains a link concept that captures the resource request. To do this, we try to retrieve the link concept that has a number of properties whose values match the values of the parameters (”DescribeMember”, ”Member”, ”John”) that are sent along with the resource request.

- Next, we need to refresh our knowledge of the Version Log. As we have seen, the Version Log keeps a list of all the versions of all the concepts of the IM. These versions are grouped per concept of the IM within the Version Log. Each concept of the IM has a single corresponding, so-called EvolutionConcept in the IM. This EvolutionConcept groups together the versions of concept it corresponds with.

- The result of the renaming change is a link in the IM that still uses the same concepts as before, although the names of all these concepts may have changed. This is where the EvolutionConcept from the previous point comes in handy. Although the concepts of the IM have changed their names, and these changes are reflected in the Version Log, the EvolutionConcepts associated with these concepts have not changed. The EvolutionConcepts form a *bridge* between the old names in the old versions it captures and the new names in the current version it captures.

- This bridge between versions also provides a bridge between the erroneous resource request (which uses the old names) and the up-to-date resource request (which uses the new names) that we are seeking.

- To limit the number of queries needed in the next step, we first retrieve all the website version numbers. The query to get this information is specified in Listing 7.10.

- We start with the first query (from Listing 7.2) that determines the EvolutionConcepts for the parameters of the erroneous resource request. We perform this query for each timestamp (=version number) found in the previous step. The query searches the a past website version of the IM in the Version Log (hence the ”BEFORE” operator) for a Link concept capturing the different nodes and parameters of our erroneous resource request. We make sure the versions of the Version Log that are found to match these nodes and parameters originate from the same version by synchronizing their version numbers.

- Now that we have determined the EvolutionConcepts we can use their identifiers as a bridge to check if the current version of the IM in the Version Log contains a definition for a link that uses the same EvolutionConcepts, we do this by performing the query of Listing 7.3.
- if the current version of the IM in the Version Log does contain a definition for a link that uses the same EvolutionConcepts, we simply return the current names for the detected concepts.
- If this is not the case, we have encountered a ”removing” change and need to proceed as described in Section 7.3.2.1.2

Listing 7.1 contains the first query that has been described. The resource request for Listing 7.3 retrieves a resource containing the Node with name ”Describe-Member” with as content the content instance of the ”Member” class referenced by the ”John” identifier.

```
// get ?x, the link
checkExistence(?x):=
// start by looking for the contentid that is the value for a property called ”
    hasValue”
hasValue(?a, ”John”) AND ofProperty(?a, hasValue) AND hasPropertyInstantiation
    (?b , ?a) AND
// that property belongs to ?b, which is in turn the value for the ”hasObject”
    property
hasTransactionTime(?b, ?c) AND hasObject(?e, ?b) AND ofProperty(?e, hasObject)
    AND hasPropertyInstantiation (?f, ?e) AND
    // that property belongs to ?f which is an ObjectChunkReference
    instanceOf(?f, ObjectChunkReference) AND hasTransactionTime(?f, ?c) AND
    // that is in turn the value for the ”hasParameter” property of ?x, the link
    hasObject(?g, ?f) AND ofProperty(?g, hasParameter) AND
        hasPropertyInstantiation (?x, ?g) AND
//?f has another property, called ”hasSubject”
 hasPropertyInstantiation (?f, ?h) AND ofProperty(?h, hasSubject) AND
// the value for that property is a class named ”Member”
hasObject(?h, ?i) AND hasID(?i, ”Member”) AND hasTransactionTime(?i, ?c) AND
    //?x, the link has another property, namely ”hasTarget”
     hasPropertyInstantiation (?x, ?k) AND ofProperty(?k, hasTarget) AND
        hasObject(?k, ?l) AND
    // which has as value a Node instance
```

```
     instanceOf (?l, Node) AND hasTransactionTime(?l, ?c) AND
          hasPropertyInstantiation (?l, ?n) AND
          // which has as a value for the "hasName" property "DescribeMember"
          ofProperty (?n, hasName) AND hasValue(?n, "DescribeMember") AND
instanceOf (?x, Link) AND hasTransactionTime(?x, ?c);
```

Listing 7.1: The Version Log query to determine whether the current version of the IM contains a link concept that matches the resource request

Listing 7.2 contains the second query that has been described. The resource request for Listing 7.3 retrieves a resource containing the Node with name ”DescribeMember” with as content the content instance of the ”Member” class referenced by the ”John” identifier. And the website transactiontime is represented by ”00”.

```
// get the evolution concepts
findLink (?d,?j,?m,?y):=
// assign the evolution concept of the node to ?m
<BEFORE(?x)>(hasValue(?n, "DescribeMember") AND ofProperty(?n, hasName)
     AND hasPropertyInstantiation(?l, ?n) AND hasTransactionTime(?l, "00") AND
     instanceOf(?l, Node) AND hasVersion(?m, ?l) AND
     hasObject(?k, ?l) AND ofProperty(?k, hasTarget) AND
          hasPropertyInstantiation (?x, ?k) AND hasTransactionTime(?x, "00") AND
      hasPropertyInstantiation (?x, ?g) AND hasObject(?g, ?f) AND
          hasTransactionTime(?f, "00") AND instanceOf(?f, ObjectChunkReference)
          AND
// assign the evolution concept of the content key to ?d
 hasPropertyInstantiation (?f, ?e) AND ofProperty(?e, hasObject) AND hasObject(?
     e, ?b) AND hasVersion(?d, ?b) AND hasTransactionTime(?b, "00") AND
      hasPropertyInstantiation (?b, ?a) AND ofProperty(?a, hasValue) AND
     hasValue(?a, "John") AND
// assign the evolution concept of the content class to ?j
 hasPropertyInstantiation (?f, ?h) AND ofProperty(?h, hasSubject) AND hasObject
     (?h, ?i) AND hasID(?i, "Member") AND hasTransactionTime(?i, "00") AND
     hasVersion(?j, ?i) AND
// assign the evolution concept of the link to ?y
instanceOf (?x, Link) AND hasVersion(?y, ?x));
```

Listing 7.2: The Version Log query to retrieve the evolution concepts of a link associated with a resource request

Listing 7.3 contains the third query that has been described. The evolution concept identifiers for the evolution concepts are: ”ec_00” for the link evolution concept, ”ec_01” for the node evolution concept, ”ec_02” for te content class evolution concept and ”ec_03” for the content key evolution concept.

```
// retrieve the parameters for the resource request
findRenamed(?y,?d,?j,?m):=
// assign the transaction time to ?y
hasVersion([ec_00], ?x) AND hasTransactionTime(?x, ?y) AND
      hasPropertyInstantiation (?x, ?a) AND ofProperty(?a, hasTarget) AND
          hasObject(?a, ?b) AND hasVersion([ec_01], ?b) AND instanceOf(?b, Node)
          AND hasTransactionTime(?b, ?y) AND
         // assign the node name to ?d
            hasPropertyInstantiation (?b, ?c) AND ofProperty(?c, hasName) AND
                hasValue(?c, ?d) AND
      hasPropertyInstantiation (?x, ?e) AND ofProperty(?e, hasParameter) AND
          hasObject(?e, ?f) AND hasTransactionTime(?f, ?y) AND instanceOf(?f,
          ObjectChunkReference) AND
          // assign the content key to ?j
            hasPropertyInstantiation (?f, ?g) AND ofProperty(?g, hasObject) AND
                hasObject(?g, ?h) AND hasVersion([ec_02], ?h) AND
                hasTransactionTime(?h, ?y) AND hasPropertyInstantiation (?h , ?i)
                AND ofProperty(?i, hasValue) AND hasValue(?i, ?j) AND
          // assign the content class name to ?m
            hasPropertyInstantiation (?f, ?k) AND ofProperty(?k, hasSubject) AND
                hasObject(?k, ?l) AND hasVersion([ec_03], ?l) AND hasID(?l, ?m)
                AND hasTransactionTime(?l, ?y);
```

Listing 7.3: The Version Log query to retrieve the resource request for a renamed link

7.3.2.1.2 Handling the ”removing resources/content” Changes

The removal of a resource or a content object invalidates the WSDM resource requests, as we mentioned in Section 6.3.2. An example of this type of change is a live-time link that originally uses the following identifiers to retrieve a webpage resource that describes a member of a fanclub:

DescribeMember This is the name of the node from the Navigational Model that is requested by the link;

Member This is the name of the class that identifies the content on the schema level;

John This is the key of the instance of the Member class that identifies the content on the data level.

Suppose that in the current version of the website IM, we have removed the link concept that groups these identifiers. This would invalidate all the resource requests that are associated with that resource request.

Note that we have again grouped together the remove changes for content objects as well as for resources, since the recovery procedure for both changes is identical.

We again make the following assumptions about the current state of the environment:

- A complete Version Log exists and the IM is internally consistent;
- The resource request is of the temporal context type ”live-time”, since this type of change does not apply to resource requests with a ”snapshot” temporal context.

We start by giving a step-by-step description of the algorithm used, followed by a more detailed description of the main query used.

Algorithm

1. As in the Previous Section, we first need to determine whether the resource request might cause an error. We do this by executing the query from Listing 7.1 that was described in the previous Section.
2. Next, we need to determine the type of change causing the resource request to be erroneous. This is done using the same query as given in Listings 7.2 and 7.3 in Section 7.3.2.1.1.
3. To limit the number of queries needed in the next step, we first retrieve all the website version numbers. The query to get this information is specified in Listing 7.10.
4. We now need to determine the version number of the version containing the most recent non-removed version of the link corresponding to the erroneous resource request. Because the resource request that is represented by this link may also have changed prior to deletion, we also need to retrieve the most recent resource request captured by the link. This is done by using the main query described in the next Paragraph. This query returns the version number and the parameters of the link concept that we seek and we perform this query once for every website version number that we have retrieved in the previous step.
5. We now have the version number and the parameters needed to create a correct resource request for the wanted resource for that version of the IM. From this information, we can create a new, correct, so-called ”snapshot”

type resource request, which will be able to retrieve the wanted information. The way in which these ”snapshot” requests are responded to by our approach is described in Section 7.3.2.1.3.

6. We can now redirect the erroneous resource request to the snapshot resource request, by which the error, that could have been caused by the erroneous resource request, is corrected.

Description of the Version Log Query

- Because the concepts that make up the link corresponding to the erroneous resource request may have been renamed prior to removal, we need to make sure that we retrieve the last valid version of the link concept preceding this removal.
- To achieve this, we use the same approach as described in the previous Section. We start by retrieving the EvolutionConcept concepts by using the query from Listing 7.2 in Section 7.3.2.1.1
- Now we have the EvolutionConcepts involved in the versions of the Link concept of the erroneous resource request. Our query from Listing 7.4 will now retrieve the most recent versions of these EvolutionConcepts that form valid links and return the version number and parameter values of these versions.

Listing 7.4 contains the full query that has been described. The evolution concept identifiers for the evolution concepts are: ”ec_00” for the link evolution concept, ”ec_01” for the node evolution concept, ”ec_02” for te content class evolution concept and ”ec_03” for the content key evolution concept. And the current website transactiontime is represented by ”00”.

```
//we retrieve the identifiers for the resource request
findRemoved(?d,?j,?m):=
<BEFORE(?x)>(hasVersion([ec_00], ?x) AND hasTransactionTime(?x, ”00”) AND
      hasPropertyInstantiation (?x, ?a) AND ofProperty(?a, hasTarget) AND
          hasObject(?a, ?b) AND hasVersion([ec_01], ?b) AND instanceOf(?b, Node)
          AND hasTransactionTime(?b, ”00”) AND
          //we assign the node name to ?d
            hasPropertyInstantiation (?b, ?c) AND ofProperty(?c, hasName) AND
                hasValue(?c, ?d) AND
      hasPropertyInstantiation (?x, ?e) AND ofProperty(?e, hasParameter) AND
          hasObject(?e, ?f) AND hasTransactionTime(?f, ”00”) AND instanceOf(?f,
          ObjectChunkReference) AND
```

```
        hasPropertyInstantiation (?f, ?g) AND ofProperty(?g, hasObject) AND
            hasObject(?g, ?h) AND hasVersion([ec_03], ?h) AND
            hasTransactionTime(?h, "00") AND
            // we assign the content key to ?j
              hasPropertyInstantiation (?h , ?i) AND ofProperty(?i, hasValue)
                  AND hasValue(?i, ?j) AND
        // we assign the content class to ?m
        hasPropertyInstantiation (?f, ?k) AND ofProperty(?k, hasSubject) AND
            hasObject(?k, ?l) AND hasVersion([ec_02], ?l) AND hasID(?l, ?m)
            AND hasTransactionTime(?l, "00"));
```

Listing 7.4: The Version Log query to retrieve the most recent resource request and version number for a link that has been removed

7.3.2.1.3 Handling The "changing content" Change

As we mentioned in Section 6.3.2, a change in the content of a resource does not invalidate the resource request. To accommodate this type of change we have defined an extension of the Link concept of the WSDM Ontology, as was described in Section 7.3.1.1. Because a change in the contents does not invalidate the resource request, the algorithm we present here does not describe a recovery process, but an extension for the resource request handling mechanism. An example of this type of change is a snapshot link that uses the following identifiers to retrieve the version identified by the version number "4" of a webpage resource that describes a member of a fanclub:

DescribeMember This is the name of the node from the Navigational Model that is requested by the link;

Member This is the name of the class that identifies the content on the schema level;

John This is the key of the instance of the Member class that identifies the content on the data level.

We make the following assumptions about the current state of the environment:

- A complete Version Log exists and the IM is internally consistent;
- The resource request is of the temporal context type "snapshot", since the handling mechanism for a resource request with a "live-time" temporal context is not different from the standard resource request handling for the WDSM Method.

Algorithm

1. Upon the point of receiving the resource request with a ”snapshot” temporal context, we need to recover the IM version indicated by the version number that is received as a part of this resource request.

2. To be able to perform this recovery, we first need to retrieve the concept versions from the Version Log that have the received version number as their version number. We do this by the simple query defined in the Change Definition Language shown in Listing 7.5.

```
retrieveConcepts (?x):=<BEFORE(?x)>(hasTransactionTime(?x, ”4”));
```

Listing 7.5: The Version Log query to retrieve all Version Log versions with a particular version number

3. Now that we have the full needed version of the IM described by the concepts of the Version Ontology, we can transform these concepts into concepts of the WSDM Ontology thus recovering the IM to the needed version.

4. To limit the number of queries needed in the next step, we retrieve all the website version numbers. The query to get this information is specified in Listing 7.10.

5. We now need to determine the correct resource request corresponding link contained in the recovered IM. To do this, we first use the query from Listing 7.2 to retrieve the evolution concepts of the link associated with our snapshot resource request.

6. We now have the evolution concepts and the version number for the link concept whose resource request parameters we wish to retrieve.
   Listing 7.6 contains the query that is able to retrieve these parameters. The evolution concept identifiers for the evolution concepts are: ”ec_00” for the link evolution concept, ”ec_01” for the node evolution concept, ”ec_02” for te content class evolution concept and ”ec_03” for the content key evolution concept. And the current website transactiontime is ”4”.

```
//we retrieve the identifiers for the resource request
findLinkVersion(?d,?j,?m):=
<BEFORE(?x)>(hasVersion([ec_00], ?x) AND hasTransactionTime(?x, ”4”)
    AND
     hasPropertyInstantiation (?x, ?a) AND ofProperty(?a, hasTarget) AND
         hasObject(?a, ?b) AND hasVersion([ec_01], ?b) AND instanceOf(?b,
         Node) AND hasTransactionTime(?b, ”4”) AND
```

```
      // we assign the node name to ?d
       hasPropertyInstantiation (?b, ?c) AND ofProperty(?c, hasName)
          AND hasValue(?c, ?d) AND
   hasPropertyInstantiation (?x, ?e) AND ofProperty(?e, hasParameter)
      AND hasObject(?e, ?f) AND hasTransactionTime(?f, "4") AND
      instanceOf(?f, ObjectChunkReference) AND
       hasPropertyInstantiation (?f, ?g) AND ofProperty(?g, hasObject)
          AND hasObject(?g, ?h) AND hasVersion([ec_03], ?h) AND
          hasTransactionTime(?h, "4") AND
          // we assign the content key to ?j
           hasPropertyInstantiation (?h , ?i) AND ofProperty(?i, hasValue
               ) AND hasValue(?i, ?j) AND
      // we assign the content class to ?m
       hasPropertyInstantiation (?f, ?k) AND ofProperty(?k, hasSubject)
          AND hasObject(?k, ?l) AND hasVersion([ec_02], ?l) AND hasID
          (?l, ?m) AND hasTransactionTime(?l, "4"));
```

Listing 7.6: The Version Log query to retrieve the resource request for a link with a certain version number

From the retrieved parameters we can now reconstruct the live-time resource request that was described by the link for the version number that was specified in the snapshot resource request.

7. The recovered version of the IM is now able to satisfy the new, live-time resource request that we have determined in the previous step.

7.3.2.2 Handling Extensions

The following sections will describe the algorithms needed for the extensions of our approach described in Section 7.3.1.

7.3.2.2.1 Handling Temporal Context Version Number Addition

As we mentioned in Section 7.3.1.1, the value for the `hasContextVersion` property may not be set by the website engineer. This Section describes the algorithm that sets these values. As described in Section 7.3.1.1, these values need to be set during the Implementation phase of the WSDM Method. Every new implementation triggers another run of this algorithm.
We make the following assumptions about the current state of the environment :

- A complete Version Log exists and the IM is internally consistent;
- A new version of the IM has just been generated.

Algorithm

1. Since the values for the `hasContextVersion` property have to be set in both the Version Log and the IM (our algorithm modifies the IM, this modification needs to be reflected in the Version Log) we need to determine the Link concepts of the IM and the versions for these Link concepts in Version log.

2. These concepts can be determined by querying the Version Log. Listing 7.7 shows the query defined in the Change Definition Language to retrieve these concepts.

```
retrieveNewSnapshotLinks(?x,?w):=
// retrieve  the  Link  version  and  its  evolution  concept
instanceOf(?x,  Link)  AND hasVersion(?w, ?x)
     // that  has  a  context  type  of  ”snapshot”
     AND ( hasPropertyInstantiation (?x,  ?y)  AND ofProperty(?y,
          hasContextType)  AND hasValue(?y, ”snapshot”))
     // and  that  does  not  have  a  hasContextVersion  property
     AND (NOT(hasPropertyInstantiation(?x,  ?z)  AND ofProperty(?x,
          hasContextVersion)))
```

Listing 7.7: The Version Log query to retrieve all the new Link concepts of the ”snapshot” temporal type

3. This query returns the current versions for the Link concepts and the evolution concepts of these versions.

4. Using this information, we now also have the version number for the current version of the IM.

5. We then add the `hasContextVersion` property to each of the found Link concepts, increasing the version number value assigned to this property with every addition. Since the Version Log is still monitoring the IM during these additions, they are also captured in the Version Log

7.3.2.2.2 Handling Permanent Deletion

When the website engineer has decided that the deletion of a resource needs to be permanent, the Link concept associated with the resource request for the resource needs to be removed from the IM and the reflection of this removal in the Version Log needs to be marked as ”deleted”. When we then attempt to recover from an erroneous resource request, we need to take this special permanent deletion into account. This type of recovery only occurs as a result of the ”removing”

change algorithm. This Section will specify an extension for that algorithm that takes into account the possibility of permanent deletion.
We make the following assumptions about the current state of the environment :

- A complete Version Log exists and the IM is internally consistent;
- The resource request is of the temporal context type ”live-time”, since this type of change does not apply to resource requests with a ”snapshot” temporal context.
- The resource request has been detected to be erroneous and our recovery mechanism has been triggered.
- This recovery mechanism has detected the ”removing” change type and the algorithm has determined the evolution concept associated with the link versions in the Version Log.

Algorithm

1. We need to determine the status of the removed link concept associated with the erroneous resource request. Since we already know the evolution concept of the versions capturing the link concept in the Version Log, we can use the query from Listing 7.8 to retrieve this status value.

```
// retrieve the status string
retrieveLinkStatus (?y):= hasVersion([ ec_id ], ?x) AND hasState(?x, ?y);
```

Listing 7.8: The Version Log query to retrieve the version representing the removal of a Link

2. The above query has provided us with the value of the `hasState` property. If this value is ”deleted”, then the link has been permanently deleted and the resource the link describes a resource request for may not be recovered, in which case we return an error page for the resource request.
3. If the value of the `hasState` property equals ”retired”, this means that the resource the link describes a resource request for may be recovered, in which case we can continue with the algorithm of Section 7.3.2.1.2 where we left off.

7.3.2.2.3 Handling Website Versioning

As we argued in Section 7.3.1.3, we needed an extension of the WSDM Ontology to be able to assign a version number to a website implementation. There are two different algorithms needed here, namely:

- An algorithm to set the version number for an implementation of the website;
- An algorithm to retrieve the different website version numbers present in the Version Log.

We will treat these algorithms in the following Paragraphs:

Setting the version number
The setting of the website version numbers happens by assigning the website version number to the `hasVersion` property of the `WebSite` instance representing the website that is the result of the application of the WSDM Method. This setting happens automatically, just before the website implementation is generated from the IM.
We make the following assumptions about the current state:

- A complete Version Log exists and the IM is internally consistent;
- The version of the IM we are presented with is ready to be transformed by the Transformation Pipeline, meaning that no further changes will be made to the IM after this algorithm has been applied.

Algorithm

1. Because an IM implements a single website, we can assume that a single instance of the WebSite class exists in the IM.
2. We retrieve this single version by using the query from Listing 7.9

```
// retrieve the version that is an instance of "WebSite"
retrieveWebsite (?x):= VersionOfIndividual (?x) AND instanceOf(?x,WebSite);
```

Listing 7.9: The Version Log query to retrieve version representing the instance of the WebSite concept

3. Now that we have a version of the WebSite instance, we can retrieve the instance of the IM itself from this version.
4. We can then add (or modify) the hasVersion property to this instance, with as value the version number of the retrieved version increased by one (the addition or modification will cause an extra version in the Version Log).

Retrieving the version numbers

Since the versions in the Version Log that correspond with the website version numbers recorded in that same Version Log are the only versions that are valid candidates for recovery, we will need to be able to retrieve these website version numbers.
We make the following assumptions about the current state:

- A complete Version Log exists and the IM is internally consistent;
- The website version numbers have been added to the IM and these additions have been recorded in the Version Log.

Algorithm

1. Because an IM implements a single website, we can assume that a single instance of the WebSite class exists in the IM.
2. We retrieve the website version numbers from the hasVersion properties recorded in the Version Log by the query in Listing 7.10

```
// get the transactiontime of the link versions
 retrieveWebsiteVersions (?y):=
<BEFORE(?x)>(instanceOf(?x, WebSite) AND hasTransactionTime(?x, ?y)
    AND
     // that have a hasVersion property with a value equal to the
          transactiontime
      hasPropertyInstantiation (?x, ?z) AND ofProperty(?z, hasVersion) AND
          hasValue(?z, ?y));
```

Listing 7.10: The Version Log query to retrieve the version numbers assigned to the WebSite instance

3. The above query retrieves all the information that we need.

7.3.2.2.4 Handling File to Link Lookup

As we described in Section 7.3.1.4 the website that is the target of our recovery approach may have been generated by the WSDM Method as a static website. Such a static website consist of a number of fixed pages in a certain format (e.g. *.html files) depending on the target platform of the implementation. Section 7.3.1.4 describes an extension we have made to Link concept that links the filenames of these files to the resource requests captured by the Links concepts of the WSDM Method. This Section provides an algorithm to handle the occurrence of errors in static websites.
We make the following assumptions about the current state of the environment :

- A complete Version Log exists and the IM is internally consistent;
- The website has been implemented as a static website;
- The resource request has been detected to be an erroneous static resource request and our recovery mechanism has been triggered.

Algorithm

1. To limit the number of queries needed in the next step, we first retrieve all the website version numbers. The query to get this information is specified in Listing 7.10.
2. To retrieve the dynamic resource request associated with the static resource request we make use of our link extension and use the filename of the static resource request in the query contained in Listing 7.11. We perform this query once for each of the website versions that we have determined in the previous step
   The Link concept we are looking for in Listing 7.11 has been associated with the static file with filename ”file001.htm”.

```
// retrieve the evolution concept and version Number of the Link
retrieveDynamicLink(?y, ?z):=
<BEFORE(?x)>hasVersion(?y, ?x) AND instanceOf(?x, Link) AND
    hasTransactionTime(?x, ?z) AND
    // that has as associated filename ” file001 .htm”.
     hasPropertyInstantiation (?x, ?w) AND ofProperty(?w, requestsPageFile
        ) AND hasValue(?w, ”file001.htm”);
```

Listing 7.11: The Version Log query to retrieve the Link concept associated with a filename

3. The query from Listing 7.11 has provided us with the version number and the evolution concept of the version of the Link concept associated with the static resource request (filename).
4. From this version number and the evolution concept we can now retrieve the dynamic resource request. Listing 7.12 shows the query to retrieve this request. The dynamic resource request we need to retrieve in Listing 7.12 is from the Link with evolutionconcept ”ec_0” and version ”4”.

```
//we retrieve the identifiers for the resource request
retrieveDynamicRequest(?d,?j,?m):=
<BEFORE(?x)>(hasVersion([ec_0], ?x) AND hasTransactionTime(?x, ”4”) AND
    hasPropertyInstantiation (?x, ?a) AND ofProperty(?a, hasTarget) AND
        hasObject(?a, ?b) AND instanceOf(?b, Node) AND hasTransactionTime
        (?b, ”4”) AND
        // we assign the node name ?d
         hasPropertyInstantiation (?b, ?c) AND ofProperty(?c, hasName)
             AND hasValue(?c, ?d) AND
    hasPropertyInstantiation (?x, ?e) AND ofProperty(?e, hasParameter)
        AND hasObject(?e, ?f) AND hasTransactionTime(?f, ”4”) AND
        instanceOf(?f, ObjectChunkReference) AND
         hasPropertyInstantiation (?f, ?g) AND ofProperty(?g, hasObject)
             AND hasObject(?g, ?h) AND hasTransactionTime(?h, ”4”) AND
            // we assign the content key to ?j
             hasPropertyInstantiation (?h , ?i) AND ofProperty(?i, hasValue)
                  AND hasValue(?i, ?j) AND
        // we assign the content class to ?m
         hasPropertyInstantiation (?f, ?k) AND ofProperty(?k, hasSubject)
             AND hasObject(?k, ?l) AND hasID(?l, ?m) AND
             hasTransactionTime(?l, ”4”));
```

Listing 7.12: The Version Log query to retrieve a link for a Resource Request

5. Now that we have retrieved the dynamic resource request associated with the erroneous static resource request, we continue with the dynamic resource request and let the normal error-correction process for dynamic websites take over and correct possible errors.

7.3.2.2.5 Handling Media Source Versioning

Section 7.3.1.5 describes an extra versioning procedure for multimedia sources. This procedure is described here as an algorithm.
We make the following assumptions about the current state of the environment :

- A complete Version Log exists and the IM is internally consistent;
- A new version of the IM has just been generated.

Algorithm

1. We need to retrieve all the newly added MultimediaConcepts that contain references to multimedia sources. But this is not all, we also need to retrieve the same type of MultimediaConcepts whose references to multimedia sources have changed in this version of the IM.
   The different types of MultimediaConcepts that maintain URL references to multimedia sources are: `Audio`, `Image`, `Resource` and `Video`.

Listing 7.13 gives an overview of the query defined in the Change Definition Language that is able to retrieve the information about these new or changed MultimediaConcepts from the version log.

```
retrieveNewMultimedia(?y, ?w):=
// retrieve the Multimedia Concept ID (?y) and EvolutionConcept (?v)
 Individual (?x) AND hasID(?x, ?y) AND hasVersion(?v,?x)
// for the Multimedia Concept that is once of the applicable subtypes
AND (instanceOf(?x, Audio) OR instanceOf(?x, Image) OR instanceOf(?x,
    Resource) OR instanceOf(?x, Video))
    // that has a certain URL value (?w)
    AND ( hasPropertyInstantiation (?x, ?z) AND ofProperty(?z, hasValue)
        AND hasValue(?z, ?w))
// but no version existed in the past
AND
// bridge by ?v
NOTBEFORE(Individual(?u) AND hasVersion(?v, ?u)
AND (instanceOf(?u, Audio) OR instanceOf(?u, Image) OR instanceOf(?u,
    Resource) OR instanceOf(?u, Video))
    // that has that same value (?w)
    AND ( hasPropertyInstantiation (?u, ?t) AND ofProperty(?t, hasValue)
        AND hasValue(?t, ?w)))
```

Listing 7.13: The Version Log query to retrieve the new multimedia concepts that have been added to the IM

2. The query from the previous step has retrieved the ID’s and the URL’s for these multimedia sources. We can use these URL’s to retrieve the multimedia files that are referenced by the multimedia concepts and create local (backup) copies of these files.

3. We now update the Version Log (using the ID’s) by changing the URL’s referencing the multimedia files to point to the local backed-up copy. Any recovery operation for these multimedia files will now reference the backed-up copy, enabling recovery operations after the original file has been deleted.

Chapter 8

Illustrative Example

This Section contains an example of the application of our approach for a very simple html-based website. Section 8.1 gives an overview of the example website that will be used throughout this Chapter. Section 8.2 describes the change process that causes a resource request for one of the resources of the website, to become invalid. And finally, Section 8.3 describes the application of our approach to recover from the invalid resource request.

8.1 Example Website

The example that we use in this Chapter is a simple html website which consists of two webpages for a fanclub website for the artist known as Joe. The two webpages of the website are the main page of the fanclub and the Member Page of the only member of the fanclub, Joe himself.

The next Sections subsequently describe the IM, The Version Log and The Implementation for this website.

8.1.1 The IM

The IM groups together the different models of the WSDM Method. The models that are the most important for our approach are the Object Chunks, the Navigational Model, the Site Structure Model and the Page Model, because the combination of these models define the resource requests for our website. Figure 8.1 shows the combination of the Navigational and Page models for our example website.

As shown in Figure 8.1 our Navigational Model consists of three Nodes, the Nodes identified by the names `DescribeClub`, `ListMembers` and `Describe`

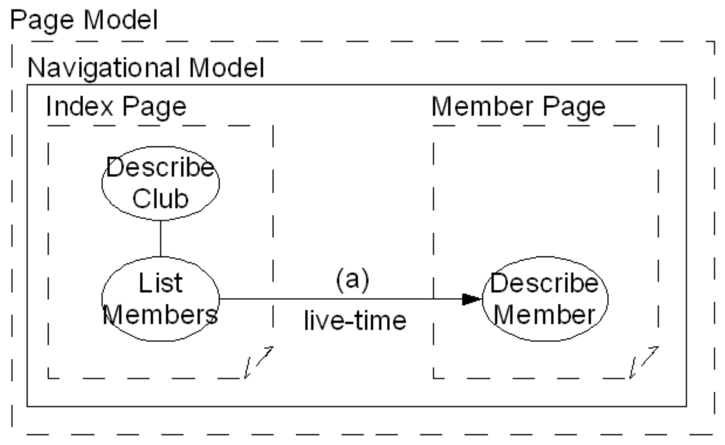


Figure 8.1: An illustration of the Navigational and Page Models for our example website.

`Member`. These Nodes are linked to Object Chunks that each fulfill a task for the website. The tasks that these Object Chunks describe are the following

- The `DescribeClub` Node is linked to the `ClubDescriptor` Object Chunk. This Object Chunk is meant to provide a description of the fanclub on the conceptual level.
- The `ListMembers` Node is linked to the `MembersList` Object Chunk. This Object Chunk is meant to list the different members of the fanclub. The Node then needs to link these members to the Member Pages.
- The `DescribeMember` Node is linked to the `MemberDescriptor` Object Chunk. This Object Chunk is meant to describe a member of the fanclub on the conceptual level.

Besides the Nodes, there are also two Links present in the Figure 8.1. The most important of these links has been labelled with (a) in Figure 8.1. This link determines the resource request that can be made to the website. Because this link links the Index Page to the Member Page, and one Index Page will link to many member Pages, the type of this link is a one-to-many link, coming from the conceptual Index Page and going to the conceptual Member Page. Because the Index Page always needs to point to the most recent version of the Member Page, the link to the Member Page will allow the Member Page to change, we specify this by giving this link the ”live-time” temporal type.
However, since WSDM Links link Nodes and not Pages, link (a) actually departs from the `ListMembers` Node and arrives in the `DescribeMember` Node.

This gives us the node that is the target of link (a), we now know that the resource request associated with (a) will contain a reference to the `DescribeMember` Node.

Now we need to determine the implementational content part of the resource request. This content part means a reference to a key value for an instantiation of the classes of the Object Chunk. Figure 8.2 visualizes the content of the `MemberDescriptor` Object Chunk. This is the Object Chunk that corresponds to the `DescribeMember` Node. The classes of Figure 8.2 describe the content

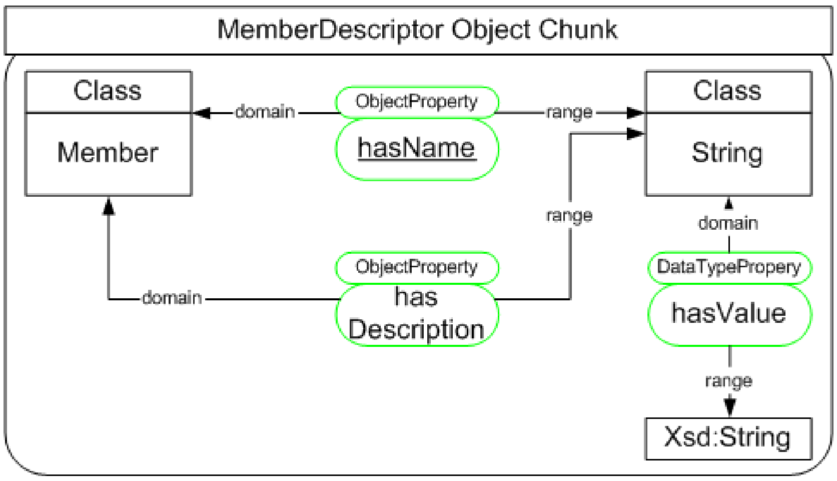


Figure 8.2: The content of the `MemberDescriptor` Object Chunk.

of the Object Chunk on the conceptual level. The implementation level for this Object Chunk is only achieved by instantiating the classes of Figure 8.2. In our implementation there is only one instance of the `Member` class because the fan-club of Joe only has one member, namely Joe. Figure 8.3 shows the instance of this Member class and the way this instance is linked to an instance of the link (a) from Figure 8.1. Figure 8.3 shows an instance of the WSDM link concept named link_0. This instance, maintains two references in its property instantiations, namely a reference to the `Node` instance identified by the ”DescribeMember” name and a reference to an `ObjectChunkReference` instance.
The `Node` instance called `nod_0` maintains a reference to the `MemberDescriptor` Object Chunk, which is in turn composed of a number of `Statement` instances. The `ObjectChunkReference` instance called `ref_0` provides a link to the components that make up a `Statement` in the Object Chunk.
As we can see in Figure 8.3, `ref_0` determines which of the statements of the Object Chunk forms the key for the content captured within that Object Chunk. There is one `Member` class and two instances of `String` concepts (subtype of

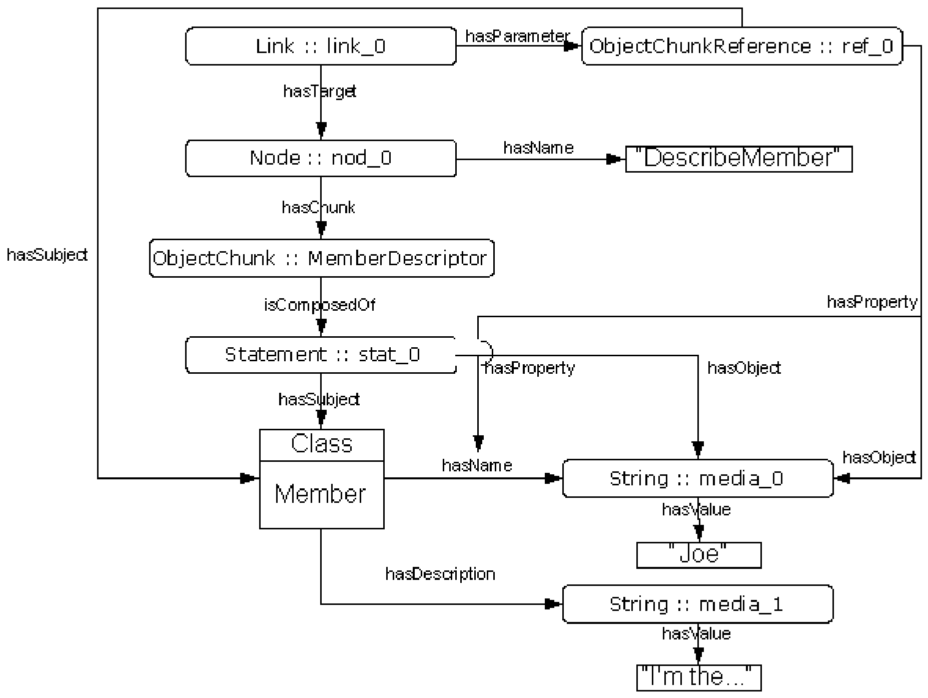


Figure 8.3: The instance of the (a) link.

the `MultimediaConcept` type) present in the Object Chunk from Figure 8.3. The values that the `String` `MultimediaConcepts` `media_0` and `media_1` are assigned in the instance of the `MemberDescriptor` `ObjectChunk` from Figure 8.3 are the following:

- The value for the `MultimediaConcept` instance `media_0` that is the Object of the hasName property is for this instance ”Joe”;
- The value for the `MultimediaConcept` instance `media_1` that is the Object of the hasDescription property is for this instance ”I’m the artist known as Joe”.

Now we have all the information needed to specify an URL that captures the resource request for the instance of the Member Page that contains the information about Joe. Figure 8.4 shows the URL that captures the resource request. As we

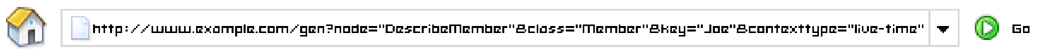

http://www.example.com/gen?node="DescribeMember"&class="Member"&key="Joe"&contexttype="live-time"

Figure 8.4: The example URL that captures the resource request.

see in Figure 8.4, the URL that captures the resource request for the Joe Member

Page is:
`http://www.example.com/gen?node="DescribeMember"&class="Member"&key="Joe"&contexttype="live-time"`

As shown in the URL, the key value used to display the information related to a member, is here the name of that member.

Now, we have described the key models of the IM that determine the resource request available for our website. The next Section will then describe how these models are reflected in the Version Log.

8.1.2 The Version Log

Since this is the first version of the IM that is being captured by the Version Log, the Version Log captures the entire IM ontology. In this Section we will concentrate on the way the Link instance from Figure 8.3 from the previous Section is captured by the Version Log. We do this because this Link instance determines the resource request that is possible for our website and this resource request will, further on, guide our recovery process.

Figure 8.5 repeats a part of the Link instance described in the Figure from the previous Section (Figure 8.3) and relates how the Link instance is captured in the Version Log. As shown in Figure 8.5 each of the concepts of the IM is represented in the Version Log by an `EvolutionConcept` which maintains a reference back to the concept (red arrow). This `EvolutionConcept` maintains a version of the individual it references by its `VersionOfIndividual` instance. This instance also maintains a reference back to the individual in the IM (green arrow), besides this reference, the `VersionOfIndividual` instance also maintains representations of the properties of that individual and a version number through its `hasTransactionTime` property. Because this is the first version of the IM and its concepts, all the versions in the Version Log have as their version numbers 0.

The next Section will describe the website, once it has been implemented.

8.1.3 The Implementation

As mentioned in the introduction, our website consists of two pages, that each fulfill a number of tasks:

The Index Page This page fulfills the tasks of describing the fanclub and listing

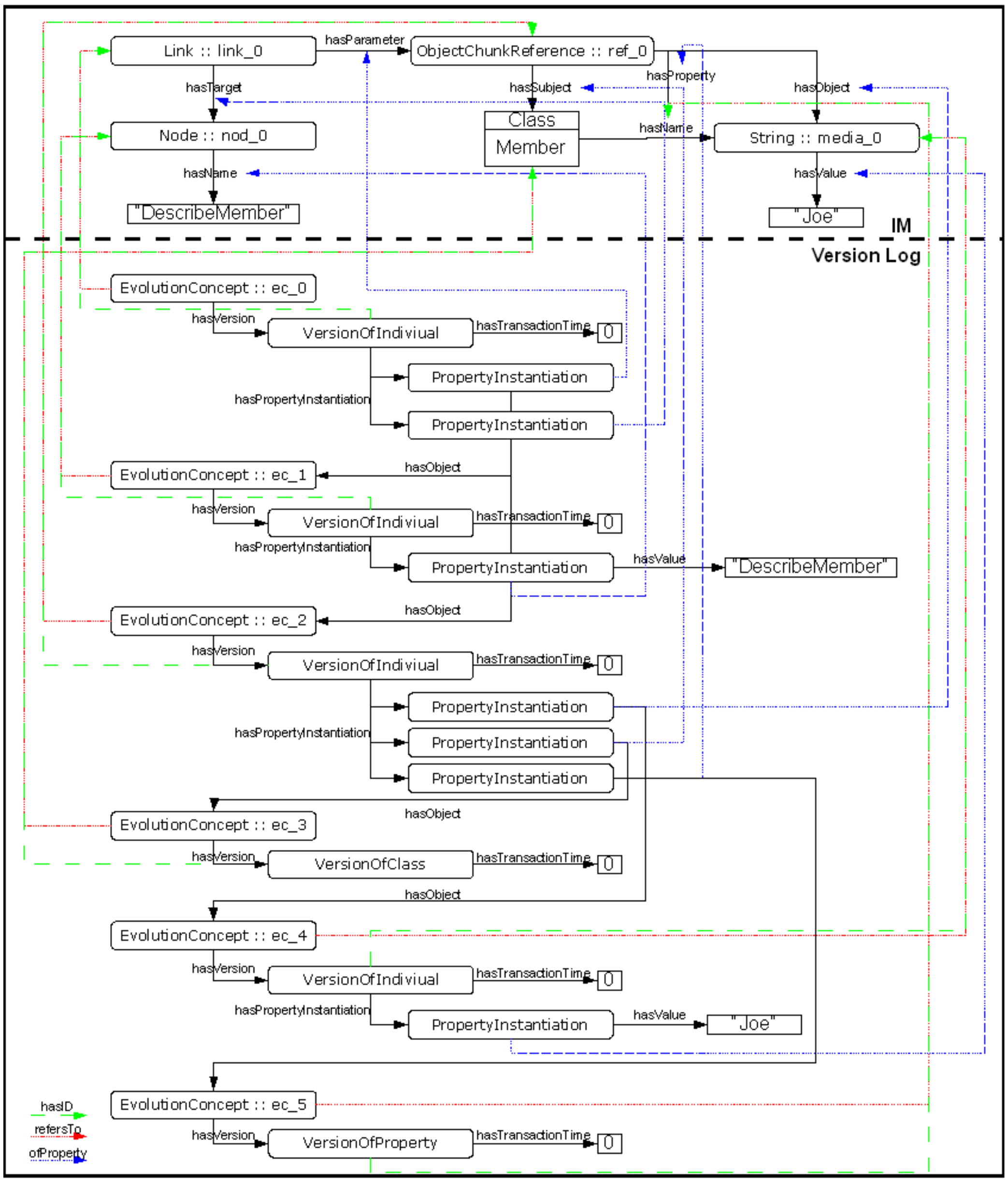
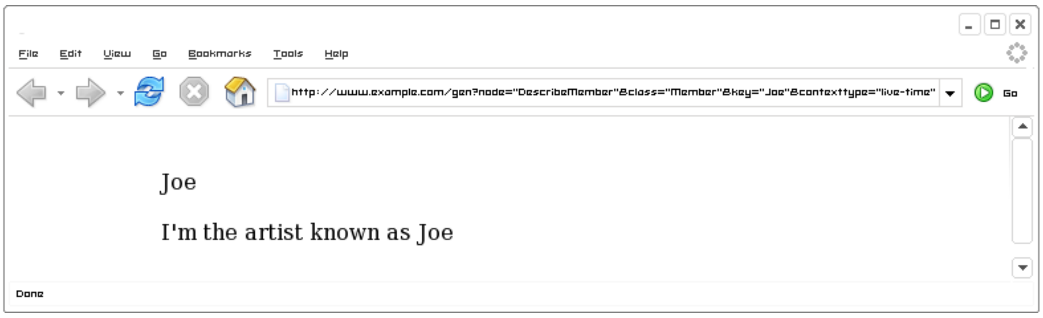


Figure 8.5: The Version Log entry for the instance of the (a) link.

the links to its members.

The Member Page This page fulfills the task of describing one member of the fanclub.

Figure 8.6 shows the implemented version of the member page.
As we can see, this page provides as content the strings that had been captured



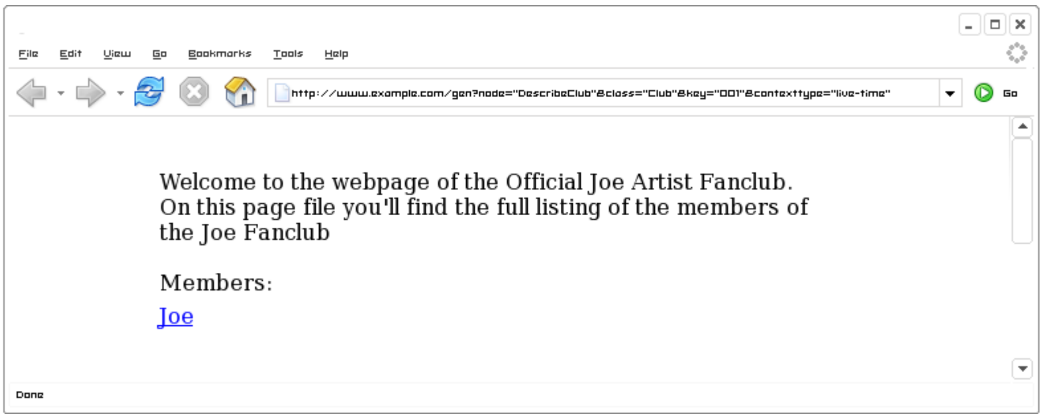
http://www.example.com/gen?node="DescribeMember"&class="Member"&key="Joe"&contexttype="live-time"

Joe

I'm the artist known as Joe

Figure 8.6: The implementation of the Member Page.

by the instance described in Figure 8.3.
Figure 8.7 shows the implemented version of the index page.
We now presume that a third party makes a link to the member page. This link



http://www.example.com/gen?node="DescribeClub"&class="Club"&key="001"&contexttype="live-time"

Welcome to the webpage of the Official Joe Artist Fanclub.
On this page file you'll find the full listing of the members of the Joe Fanclub

Members:
Joe

Figure 8.7: The implementation of the Index Page.

then uses the URL:
`http://www.example.com/gen?node="DescribeMember"`
`&class="Member"&key="Joe"&contexttype="live-time"`
to capture the resource request for the page.

The next section shows the effects of applying some changes to the website.

8.2 Example Changes

The next two Subsections will each describe a change to the website and show the effects of this change on the models of the IM and the Version Log.

8.2.1 Change 1: Renaming

8.2.1.1 The Change

Joe has decided it's time for a change. In a move to increase the popularity of his fanclub, Joe has decided to change his name into to the ”the artist formerly known as Joe” (TAFKAJ). Of course this means that the this change in name needs to be reflected in the webpages of the fanclub.

Because the name of the member is used as a key value in the resource request for the Member Page instance, a change in this name changes the resource request needed to retrieve this page resource.

The next Sections will first describe the result of this change in the IM, followed by the resulting modifications in the Version Log. The implementation will not be reviewed here since the effects of the change are too obvious to mention.

8.2.1.2 The Changed models

Figure 8.8 shows the changed instance for the Link. As the Figure shows, little has changed. However, since the string that has changed was the key value for the resource request, this simple change invalidates the previous resource request that was used by the third party to link to our web page. The new URL that is now needed to reach the Member Page instance is the following:
`http://www.example.com/gen?node="DescribeMember"&class ="Member"&key="TAFKAJ"&contexttype="live-time"`

In the next Section we will see how the change is reflected in the Version Log.

8.2.1.3 The Changed Version Log

Figure 8.9 shows the changes that have happened to the Version Log. To provide more clarity we have left out a number of elements that were present in the initial Figure (Figure 8.5 in Section 8.1.2). Figure 8.9 shows that the simple change from the previous section causes the addition of a single version to the Version Log, when the new IM is generated. This new version has as a version number the number 2.

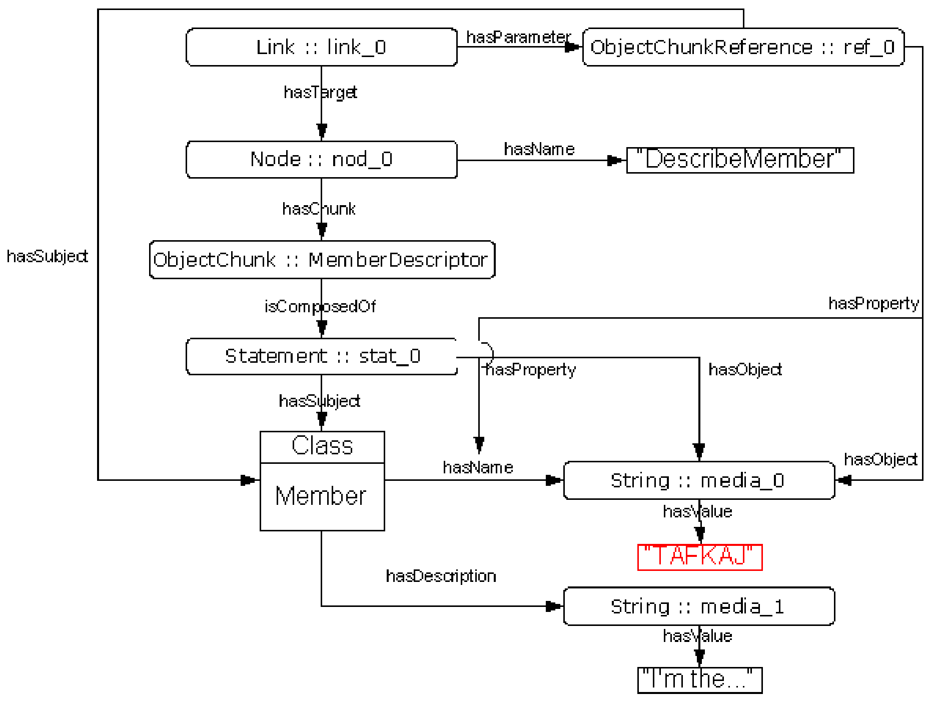


Figure 8.8: The changed instance for the Link.

Note that the Version Log still maintain the previous version of the value for the String instance, namely "Joe".

The next Section describes another type of change that is performed on the website that is the target of our approach.

8.2.2 Change 2: Removing

8.2.2.1 The Change

Because The Artist Formerly Known As Joe has been unsuccessful in his attempt to attract more members to his website, he has decided to remove the list of fanclub members from his website. This change will effectively remove every instance of the member page and the conceptual Member Page from the website.

Since the member page resource has been removed from the website, this means that the resource requests for this resource have become invalid.

The following Sections will first describe the result of this change applied to the IM, followed by the resulting entries in the Version Log.

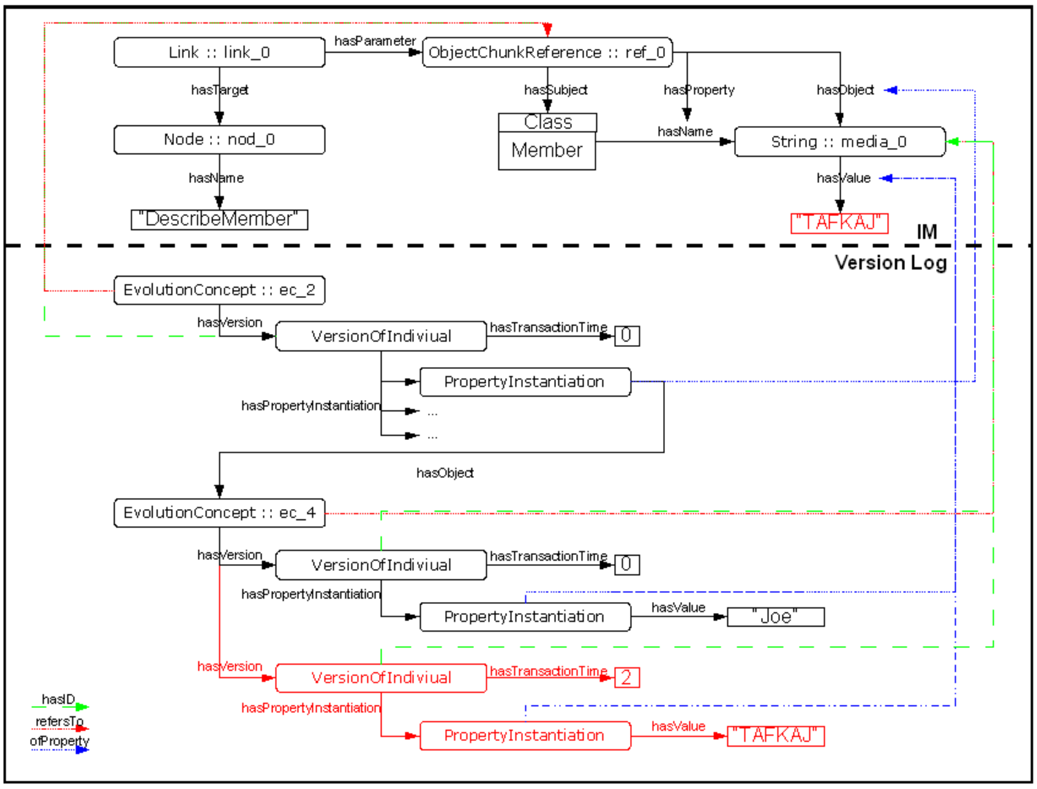


Figure 8.9: The Version Log entry for the changed instance of the Link.

8.2.2.2 The Changed models

As a result of this change, the page member instance and the link instances have been removed from the Page and Navigational Models. Figure 8.10 shows the changed version of Figure 8.1 which contains the Page and Navigational Models from the IM. The elements that have been removed from the Page and Navigational Model have been greyed out in Figure 8.10. The removal of the Member page also means the removal of the links and nodes associated with this page. In this case, the node that was the source of link (a) (the `ListMembers` Node) has also been removed because the task that was fulfilled by this node was no longer required. However, in other circumstances it may be that the node remains and only the link is removed.

The next Section will show the effects of this change on the Version Log.

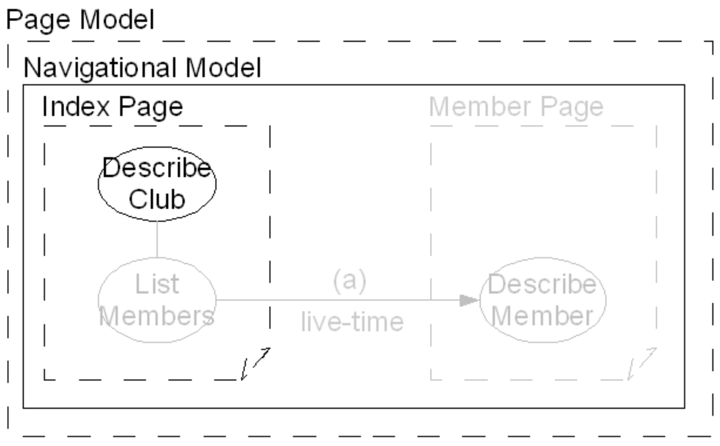


Figure 8.10: An illustration of the changed Navigational and Page Models for our example website.

8.2.2.3 The Changed Version Log

Although all the concepts that have been removed from the IM (the concepts that are related to the removed elements of Figure 8.10) cause entries in the Version Log, Figure 8.11 only shows the changes as a result from the removal of link (a). Each of the `EvolutionConcepts` of the original Version Log has received an additional version (pictured in red) representing the removal of these concepts. Because the original concepts of the Link instance have been removed, the `hasID`, the `refersTo` and the `ofProperty` properties now are no more than strings identifying the removed concepts. We review these strings for a better understanding of Figure 8.11:

- ”link0” refers to the `Link` instance that has been removed;
- ”ref_0” refers to the `ObjectChunkReference` instance (parameter of the link) that has been removed;
- ”nod_0” refers to the `Node` instance (target of the link) that has been removed;
- ”media_0” refers to the `String` (`MultiMediaConcept`) instance (object of ref_0) that has been removed.

Note that each of the newly created versions has a `hasState` property with a value of ”retired”. This string indicates the removal of the concept associated with the version and it indicates that this type of removal allows recovery. If the website engineer wishes to prevent the recovery of the resource, the `hasState`

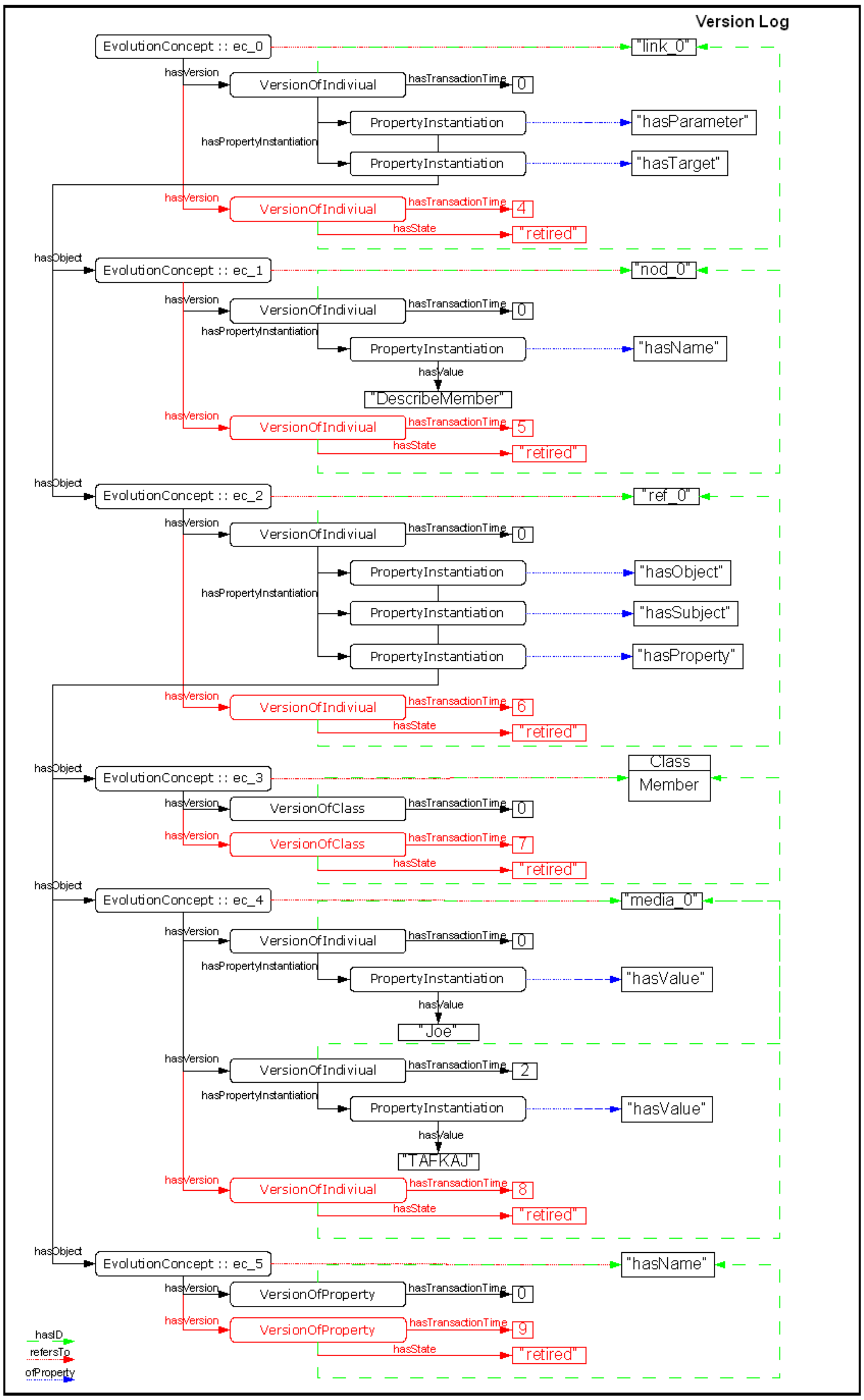


Figure 8.11: The Version Log entry for the removed instance of the Link.

property of the removed Link defining the resource request for the resource should be set to ”deleted”.

The next Section will illustrate the recovery process.

8.3 Example Approach Application

In Section 8.1.3 we described the creation of a third-party link using the URL (`http://www.example.com/gen?node="DescribeMember"&class="Member"&key="Joe"&contexttype="live-time"`) to capture the resource request.

We explain how this resource request is treated step-by-step in the following Sections.

8.3.1 Step 1: Check the existence

We start by checking if our error-correction approach needs to be applied. We do this by executing the query from Listing 7.1. This query attempts to match the identifiers of the resource request to a link concept contained in the version of the Version Log that captures the current version of the IM.
So the query attempts to match the identifiers of the resource request:

- ”DescribeMember”
- ”Member”
- ”Joe”

To the identifiers of any of the link concepts that have been captured in the current version of the Version Log.
In our example, there are no more unremoved link concepts present in the Version Log. All the link concepts were removed from the IM as a result from the removing change from Section 8.2.2.
Because the query is unable to return any matching links for the identifiers of the resource request, our error-correction mechanism is triggered.

8.3.2 Step 2: Retrieving the website versions

Now, we need to retrieve the website versions. Because the changes presented in the Section 8.2 and the initial version of the website from Section 8.1 each represent a complete iteration of the WSDM method, each of these versions are website versions and the query from Listing 7.10 will have returned the following website version numbers: 1, 3 and 33. The origins of these numbers are the following:

- 1: this is the website version number of the very first implementation of the website (which had as version number 0 + 1 for the addition of the version number property);
- 3: this is the website version number after the renaming change. The renaming change itself had been assigned the version number 2. We add one for the additional version that was caused by modifying the website version number in the IM;
- 33: this website version number is an arbitrary choice because, in the previous Section, we have only illustrated the removal of a limited number of concepts.

8.3.3 Step 3: Finding the link

By ”Finding the link”, we mean retrieving the evolution concepts from the Version Log that are associated with the concepts that together form or used to form the erroneous resource request that we have received. To find these evolution concepts, we execute the query from Listing 7.2. This query will attempt to match the identifiers of the resource request that we have received with the identifiers present in any of the website versions present in the Version Log.

For our example, this means that the query looks among the Website Versions for a version that contains a link concept that links together evolution concepts that have had the following identifier values;

- ”DescribeMember”
- ”Member”
- ”Joe”

The query is executed for each one of the website version numbers until a match has been found. Because we wish to retrieve the most recent match possible, we start with the Highest website version numbers first. Because step 1 has already shown that the current website version (with version number 33) does not contain a matching value, we start our search at the website version with version number 3.

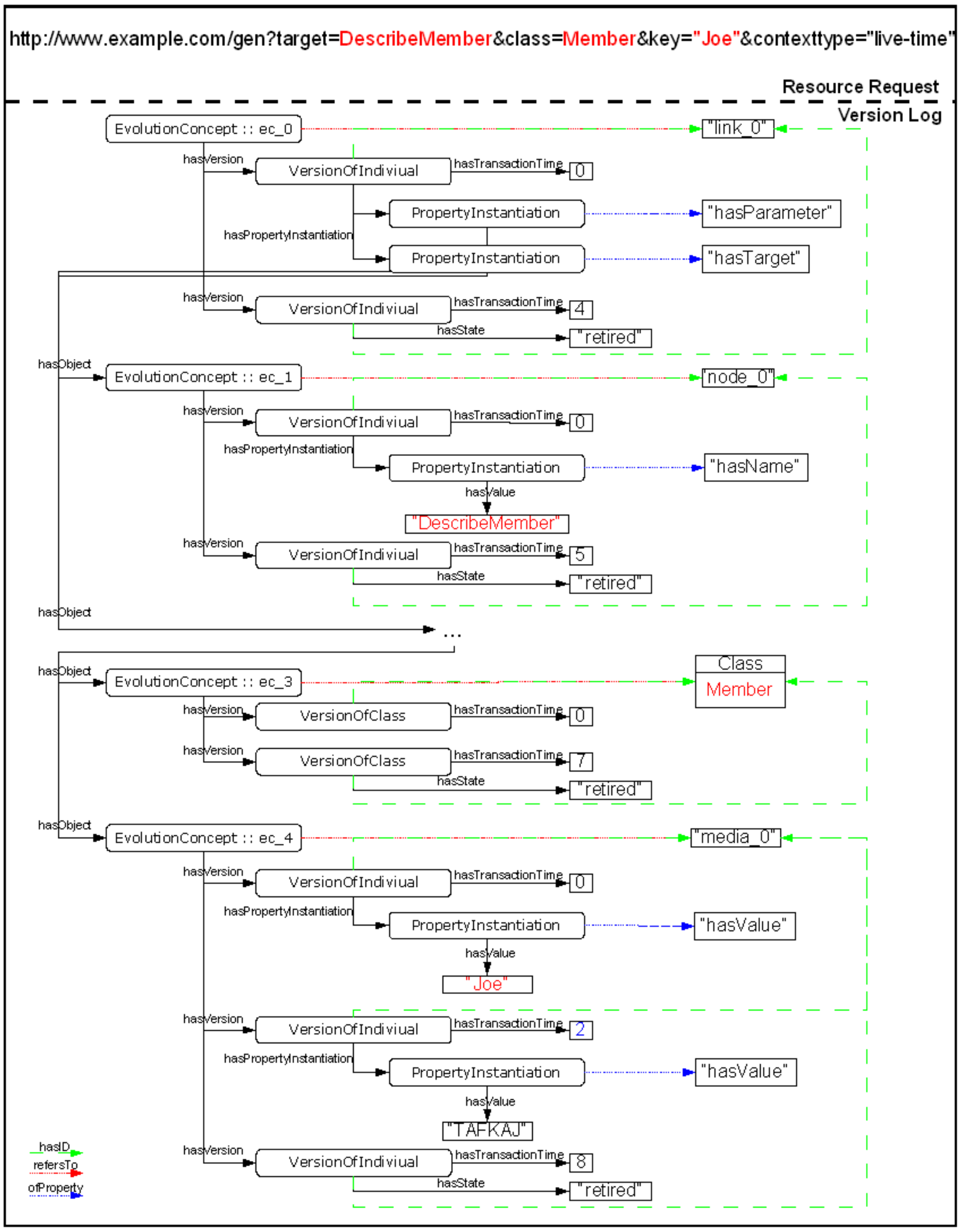


Figure 8.12: The identification of the identifiers in the Version Log.

Figure 8.12 shows the evolution concepts in the Version Log that we are seeking. We must note that none of the versions shown in Figure 8.12 actually has as version number the version number 3.
This is because the Figure 8.12 only shows the *explicit* versions of the Version Log, that have been physically stored in the Version Log. What we are missing, are the *implicit* versions. These implicit versions are implied by the explicit versions that are present. A single explicit version for an evolution concept, implies that the concept that is captured by that evolution concept remains the same as that explicit version (=unchanged) until a new explicit version is recorded in the Version Log.
This means that the Version Log shown in Figure 8.12 actually does contain versions with as version number the number 3. Take for example the evolution concept ec_0 from Figure 8.12. This evolution concept has two explicit versions, with version numbers 0 and 4. This means that from version 0 to 3, the concept represented by the evolution concept ec_0 remains unchanged (=equal to the version with version number 0).
When we now look at the evolution concept ec_4 at the bottom of Figure 8.12, we notice that it has the value ”TAFKAJ” from version number 2 to version number 7. This means that the evolution concept cannot be matched to the value ”Joe” for the website version with version number 3.
This means we need to continue with the next website version number, namely the number 1. This time the value ”Joe” does match up with the value ”Joe” from version number 0 to 1. The same case applies for the other identifiers of the resource request. The matches for these values are marked in red in Figure 8.12. We are finally able to match the following evolution concepts to the identifiers of the resource request:

- ”DescribeMember”: the name of the node finds a match for the evolution concept with id ec_1;
- ”Member”: the classname of the content finds a match for the evolution concept with id ec_3;
- ”Joe”: the content key finds a match for the evolution concept with id ec_3.

All these evolution concepts are connected together by a link concept versioned by the evolution concept with id ec_0.

8.3.4 Step 4: Find Renamed

In this step, we attempt to retrieve the renamed identifiers for the link concept that we have retrieved in the previous step. A renaming change in the IM does not affect the evolution concepts that capture these identifiers. Since we have retrieved

these evolution concepts in the previous step, we can now use these evolution concepts as input for the query from Listing 7.3. This query attempts to find a combination of the evolution concepts that have been provided in the current version of the Version Log.
If such a combination can be found, this means that a simple renaming change has taken place, and we can then create a resource request that reflects this rename from the information returned by the query.
As we can clearly see from the Figure 8.12, there are no longer connections present among the most recent versions of the evolution concepts ec_0, ec_1, ec_3 and ec_4. This means that another type of change must be the cause of our erroneous resource request, namely the removing change.

8.3.5 Step 5: Retrieve link status

Because a removing type of change has been detected, we need to make sure that we are allowed recover the resource that is requested by the erroneous resource request. to do this, we simply use the query from Listing 7.8. We provide the evolution concept id of the link concept, namely ec_0 as input for this query.
The query then retrieves the current version of this evolution concept from the Version Log and returns the value for its `hasState` property. As we see from Figure 8.12, the last explicit version of ec_0 has as version number, number 4 and has as a value for its `hasState` property the value ”retired”. Because this string value means that we may recover this link concept, we can continue on to the next step.

8.3.6 Step 6: Find removed

What we now need, is the most recent version of the link we found in Step 3, prior to its deletion. To find this version, we use the query from Listing 7.4. This query performs the same actions as the query from Step 4, except that this time, we don’t attempt to retrieve the link from the current version of the Version Log, but from all the past website versions of the Version Log.
To do this, we iterate over the website versions we retrieved in Step 2. Again, we start with the most recent version preceding the current version.
This time, we are lucky, the first website version that we try, namely the website version with version number 3, contains a combination of all evolution concepts we had found in Step 3, with the identifiers needed to create a resource request.
All that is left now, is to determine the resource request that is associated with the link in website version 3. The identifiers that determine the parameters for these links in the website version 3 have remained the same as in version 0, except for the value of the identifier captured by the versions in `ec_4`, which has changed

from ”Joe” to ”TAFKAJ”. This means that the resource request for the website version 3 is:
`http://www.example.com/gen?node="DescribeMember"&class ="Member"&key="TAFKAJ"&contexttype="live-time"`

8.3.7 Step 7: Return the snapshot resource request

Because the new resource request that we have determined is only valid in version 3, we need to make sure that our resource request retrieves that version of the IM. We can do this by changing the type of our resource request from a live-time resource request to a snapshot type resource request. This results in the following URL for the resource request:
`http://www.example.com/gen?node="DescribeMember"&class ="Member"&key="TAFKAJ"&contexttype="snapshot"&context version=3`
We now redirect the original resource request to our new resource request:
We redirect:
`http://www.example.com/gen?node="DescribeMember"&class ="Member"&key="Joe"&contexttype="live-time"`
To:
`http://www.example.com/gen?node="DescribeMember"&class ="Member"&key="TAFKAJ"&contexttype="snapshot"&context version=3`
This new resource request is then handled by the algorithm that handles the ”changing content/resource” change. We continue with this algorithm in the next step.

8.3.8 Step 8: Retrieve concepts

Through the resource request, we have received the version number of the version that should be recovered, namely the version number 3. To do this recovery, we retrieve all versions, including the implied versions with version number 3 from the Version Log by using the query from Listing 7.5.

8.3.9 Step 9: Transform Concepts

The next step is to transform the result of the previous step back into the ontology that causes the entries in the Version Log. This is possible because the Version Log has captured the entire ontology, keeping track of the instances, classes and properties of the concepts in the IM ontology that caused the Version Log entries. For example, the `VersionOfIndividual` instances from Figure 8.12

are transformed back into instances of the classes that their `instanceOf` properties (these properties are not shown in Figure 8.12) indicate.

8.3.10 Step 10 & 11: Retrieve website versions and Find Link

These Steps are a part of the algorithm to handle snapshot links, but they are exact duplicates of Steps 2 and 3 of this Section, so we will not repeat them here.

8.3.11 Step 12: Find link version

In this step we basicly handle renaming changes for snapshot type resource requests. Because the identifiers that are part of the snapshot resource request may have changed, we need to make sure that the resource request part of the snapshot resource request is still valid in the version that is requested by the contextversion number. This is done by the query from Listing 7.6. The query retrieves the link with the evolution concepts that were retrieved in Step 11 from the Version Log version with the version number equal to the version requested by the snapshot link.

8.3.12 Step 13: Return the live-time resource request

Because we now have an IM representing the version needed from Step 9 and a resource request that is valid for that recovered IM from Step 12, we can simply transform the snapshot resource request for the original IM into a live-time resource request for the recovered IM. By doing this, we allow the resource request to be treated as a normal, live-time resource request for an adapted IM. We would then return the resource request:
`http://www.example.com/gen?node="DescribeMember"&class ="Member"&key="TAFKAJ"&contexttype="live-time"`
That is to be used with the recovered IM.

Chapter 9

Proof Of Concept

In this Section, we will describe a small application that was developed as a proof of concept for our approach.
This application has been implemented as two plug-ins for the Protégé Ontology Editor [1]. Protégé is a tool for the creation and editing of ontologies. The first Protégé plugin created as a part of our application focuses on the creation and visualization of the Version Log Component of our approach (see Section 7.2.2.4) for ontologies developed in Protégé. The second Protégé plugin implemented provides an interface to apply the algorithms of our Approach to that Version Log.

We illustrate the developed application using the example presented in Chapter 8. Figure 9.1 shows the Protégé Ontology Editor, with the ontology corresponding to the IM from the example of Chapter 8 already opened. Figure 9.1 shows the details for the media_0 concept of the IM, as indicated on the Figure, this concept has as value ”Joe”, which is the key value for the content of the Member Page from the example from Chapter 8. We will focus on this concept for the description of our plugins in the next Sections.

The next sections each describe one of the plugins of our application and their usage.

- Section 9.1 describes the Version Log plugin and the way it provides versioning for our example;
- Section 9.2 describes the Request Recovery plugin and how it provides recovery for our example.

[1]The Protégé Ontology Editor, see `http://protege.stanford.edu/`

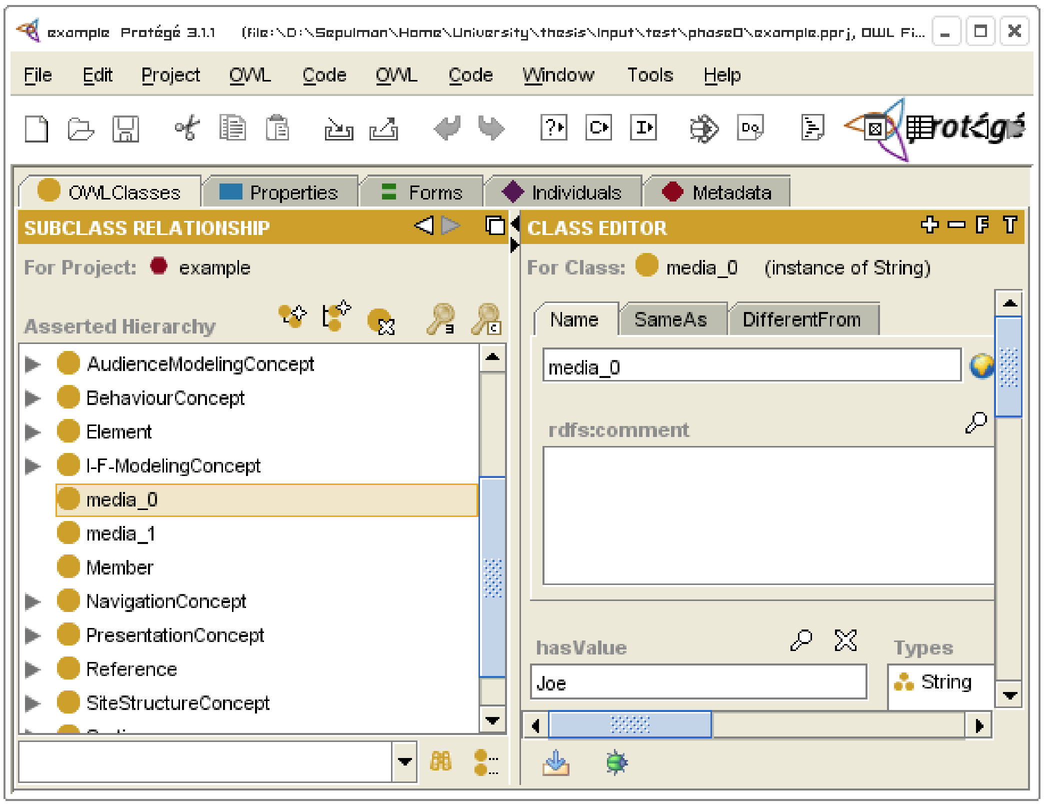
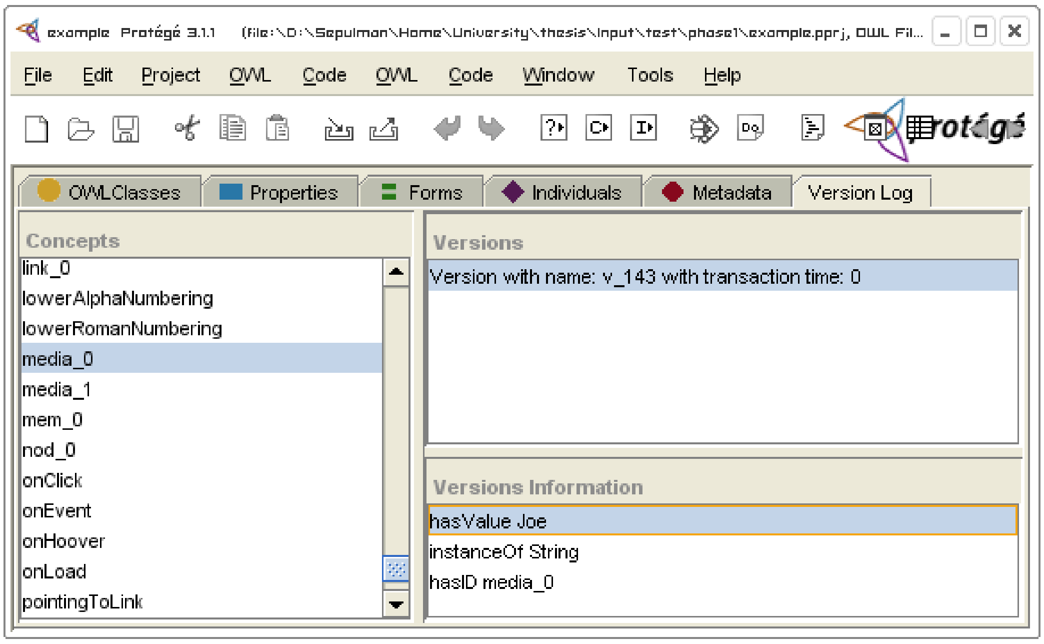

example Protégé 3.1.1 (file:\D:\Sepulman\Home\University\thesis\input\test\phase0\example.pprj, OWL Fi...
File Edit Project OWL Code OWL Code Window Tools Help
OWLClasses Properties Forms Individuals Metadata
SUBCLASS RELATIONSHIP
For Project: example
Asserted Hierarchy
AudienceModelingConcept
BehaviourConcept
Element
I-F-ModelingConcept
media_0
media_1
Member
NavigationConcept
PresentationConcept
Reference
SiteStructureConcept
CLASS EDITOR
For Class: media_0 (instance of String)
Name SameAs DifferentFrom
media_0
rdfs:comment
hasValue
Joe
Types
String

Figure 9.1: The Protégé Ontology Editor with the IM ontology opened.

9.1 Versioning

This section describes the versioning of the IM, achieved by our Version Log plugin for the Protégé Ontology Editor. This plugin creates and visualizes the Version Log.
This Version Log is created by monitoring the modifications the ontology engineer does to the ontology loaded in Protégé. Figure 9.2 shows the Version Log plugin visualizing the Version Log for the IM that was shown in Figure 9.1. In Figure 9.2, we again focus on the media_0 concept, and show how this concept is recorded in the Version Log.
The changes that are done to the IM using the Protégé Ontology Editor will then cause new versions to be added to the Version Log that reflect the results of these changes.
The Version Log is kept by our plugin as an OWL file, using the Version Ontology that was described in Section 4.3.8. To create and maintain this OWL file, our



link_0
lowerAlphaNumbering
lowerRomanNumbering
media_0
media_1
mem_0
nod_0
onClick
onEvent
onHoover
onLoad
pointingToLink

Version with name: v_143 with transaction time: 0

hasValue Joe
instanceOf String
hasID media_0

Figure 9.2: The Version Log plugin with the initial version of the IM.

plugin makes use of the Jena[2] Framework for Java.

The next Sections will each describe how the changes from the example of Chapter 8 are performed using Protégé and how these changes are reflected in the Version Log.

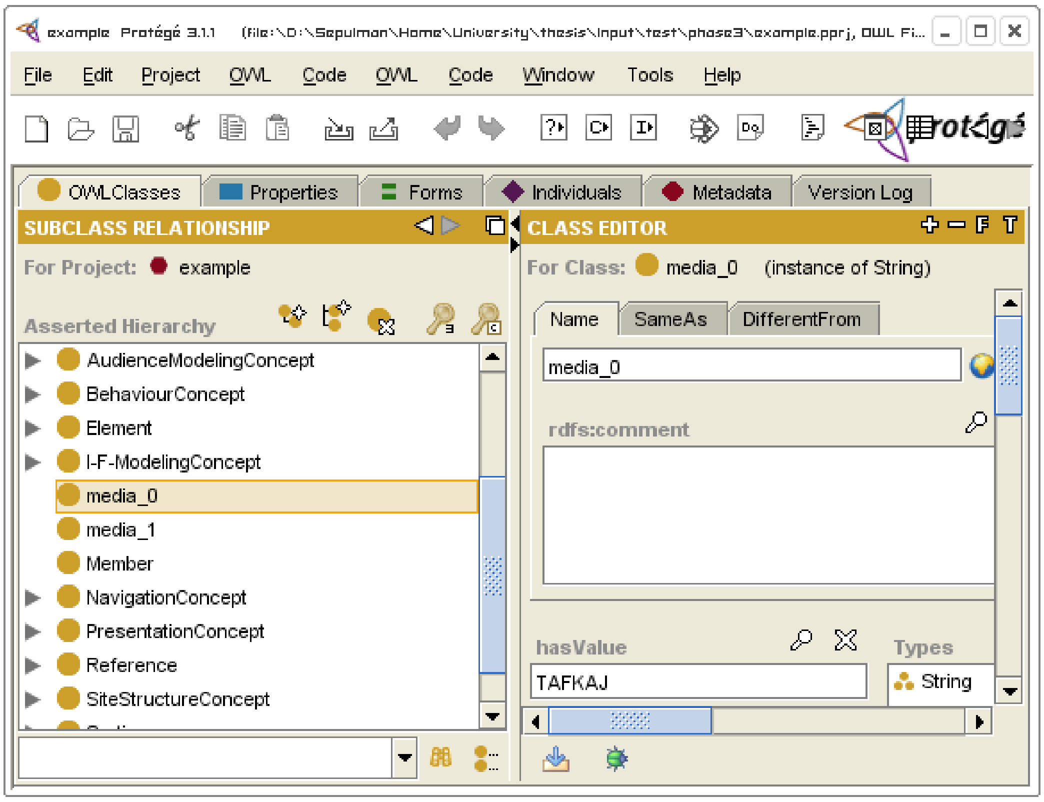
9.1.1 Change 1: Renaming

This change from Chapter 8 causes the key for the content of the Member Page to change from ”Joe” into ”TAFKAJ” (The Artist Formerly Known As Joe). Figure 9.3 shows that this change is performed on the IM using Protégé.

This change triggers the addition of a new version to the Version Log by the Version Log plugin. Figure 9.4 shows the Version Log plugin visualizing the new version (v_466 with transactiontime 3) that was added to the Version Log.

Because the Version Log also maintains the previous versions, these versions can be used as a basis for IM recovery.

[2]Jena A Semantic Web Framework for Java, see `http://jena.sourceforge.net/`


example Protégé 3.1.1 (file:\D:\Sepulman\Home\University\thesis\input\test\phase3\example.pprj, OWL Fi...
File Edit Project OWL Code OWL Code Window Tools Help
OWLClasses Properties Forms Individuals Metadata Version Log
SUBCLASS RELATIONSHIP
CLASS EDITOR
Asserted Hierarchy
Name SameAs DifferentFrom
rdfs:comment
hasValue
Types

For Project: example

AudienceModelingConcept
BehaviourConcept
Element
I-F-ModelingConcept
media_0
media_1
Member
NavigationConcept
PresentationConcept
Reference
SiteStructureConcept

For Class: media_0 (instance of String)

Name: media_0

hasValue: TAFKAJ

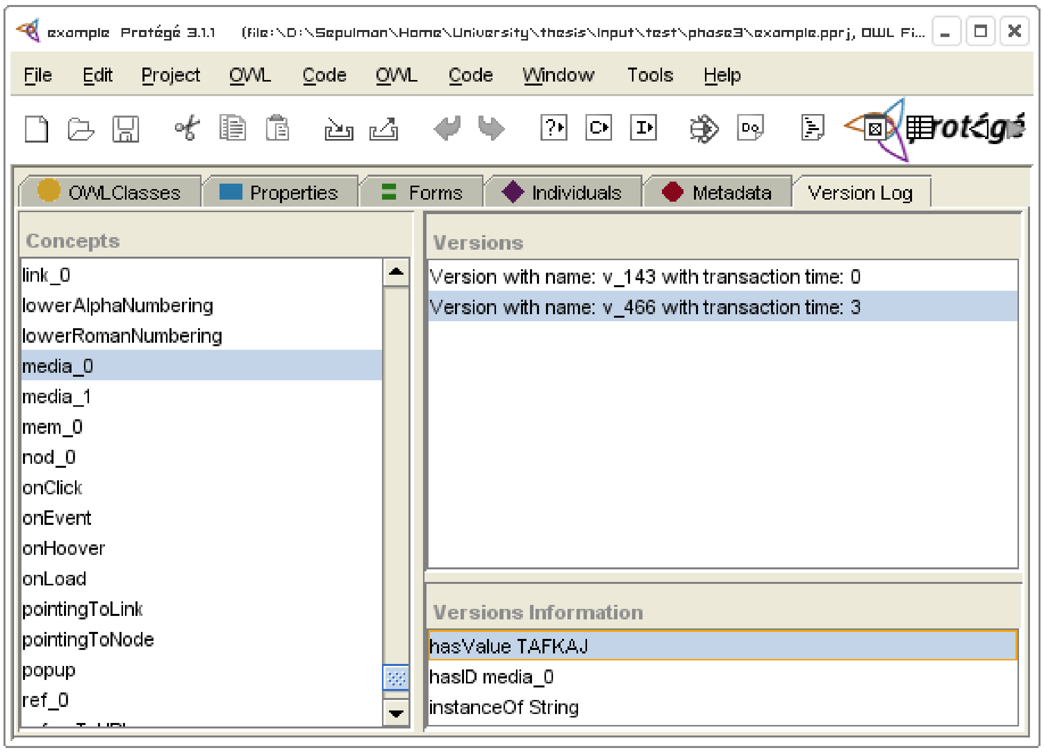
Types: String

Figure 9.3: We change the value for the concept from ”Joe” to ”TAFKAJ”.

9.1.2 Change 2: Removing

This Section illustrates how our application records the removing change from the example of Chapter 8. In this change, all the concepts related to the Member Page are removed from this IM, this also includes the media_0 and link concepts that have been captured in the example IM in this Section. Figure 9.5 shows the effects of this removal on the Version Log. Each of the concepts that have been removed receive an additional version representing the removal of these concepts. In Figure 9.5 we see such a version for the removal of the media_0 concept from the IM ontology.
We can see in Figure 9.5 that the last version of the media_0 concept has as value for the ”hasState” property the string ”retired”, indicating the removal of this concept.

Now that we have illustrated the Version Log plugin and the way it provides versioning for our approach, we will continue with the recovery aspect of our approach in the next Section.


example Protégé 3.1.1 (file:\D:\Sepulman\Home\University\thesis\Input\test\phase3\example.pprj, OWL Fi...
File Edit Project OWL Code OWL Code Window Tools Help
OWLClasses Properties Forms Individuals Metadata Version Log

Concepts

link_0
lowerAlphaNumbering
lowerRomanNumbering
media_0
media_1
mem_0
nod_0
onClick
onEvent
onHoover
onLoad
pointingToLink
pointingToNode
popup
ref_0

Versions

Version with name: v_143 with transaction time: 0
Version with name: v_466 with transaction time: 3

Versions Information

hasValue TAFKAJ
hasID media_0
instanceOf String

Figure 9.4: The change from ”Joe” to ”TAFKAJ” is reflected in the Version Log.

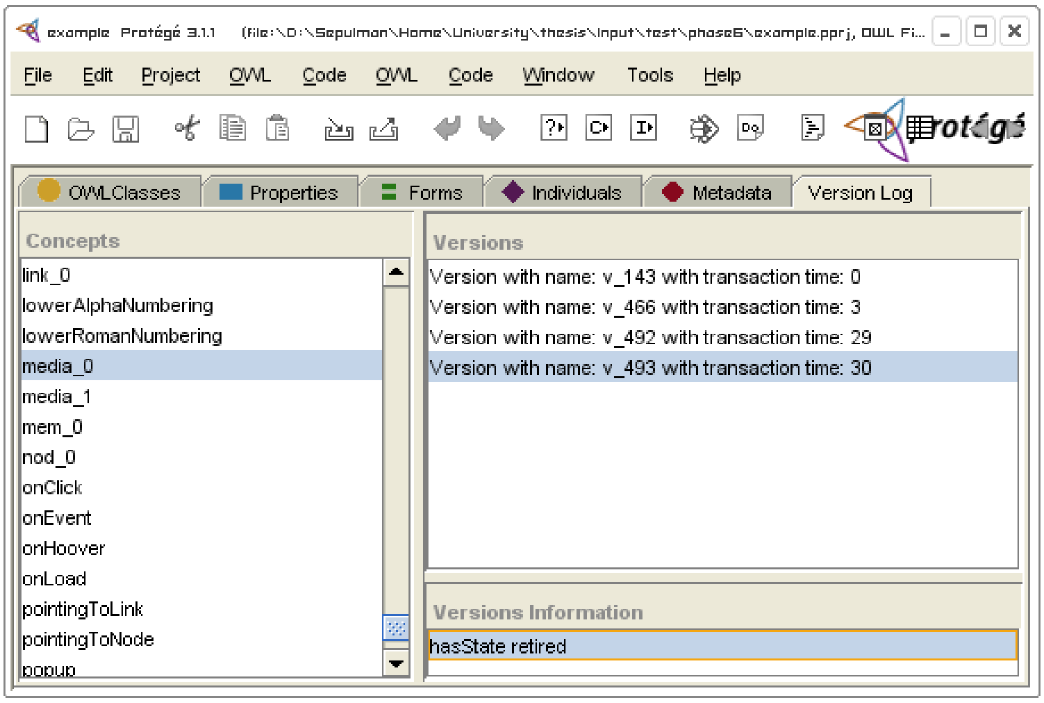
9.2 Recovering

The Request Recovery plugin provides the functionality that would be useful in the case of website error-correction. The next Sections each describe a part of the functionality provided:

- Section 9.2.1 illustrates the approach used to add Website Version numbers to the IM;
- Section 9.2.2 shows how our plugin can detect and correct an erroneous result request from an url sent to our website;
- Section 9.2.3 describes how our plugin restores a previous version of the IM using the Version Log.

This plugin makes use of the following components:

- The Jena Framework is used to access the Version Log in order to reach the functionality described in Sections 9.2.1 and 9.2.3. Additionally, Jena is used as a means to save the recovered IM from Section 9.2.3 to an OWL file.


example Protégé 3.1.1 (file:\D:\Sepulman\Home\University\thesis\input\test\phase6\example.pprj, OWL Fi...
File Edit Project OWL Code OWL Code Window Tools Help
OWLClasses Properties Forms Individuals Metadata Version Log

Concepts

link_0
lowerAlphaNumbering
lowerRomanNumbering
media_0
media_1
mem_0
nod_0
onClick
onEvent
onHoover
onLoad
pointingToLink
pointingToNode
popup

Versions

Version with name: v_143 with transaction time: 0
Version with name: v_466 with transaction time: 3
Version with name: v_492 with transaction time: 29
Version with name: v_493 with transaction time: 30

Versions Information

hasState retired

Figure 9.5: The removal of the concepts is recorded in the Version Log.

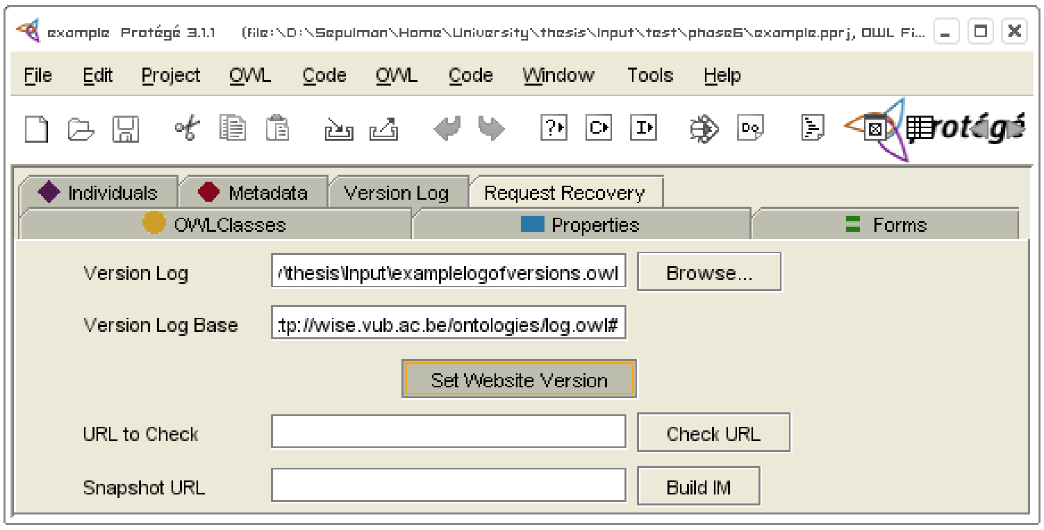
- The Change Definition Language (from Section 4.3.9) is used to query the Version Log in order to reach the functionality described in all three Sections. We have implemented this Change Definition Language using the components listed here, including the Jena Framework.
- Antlr[3] is used to generate the parser for the Change Definition Language from a grammatical description for this language.
- RDQL[4] is the RDF Query language that is used to aid in the evaluation of the statements from the Change Definition Language. RDQL is used through the RDQL api provided by Jena.

9.2.1 Website Versioning

Our approach requires that a version number is added for each iteration of the Implementation phase of the WSDM Method. This version number is added programmatically to the IM by the Request Recovery plugin.
Figure 9.6 shows the interface of the Request Recovery plugin. We first need to

[3]Antlr Parser Generator, see `http://www.antlr.org/`
[4]RDQL - A Query Language for RDF, see `http://www.w3.org/Submission/RDQL/`

input the location of the Version Log file, together with the base URI of this Version Log file. By using the "Set Website Version" button when the IM has reached a state ready for implementation, the website engineer can automatically add the current version number of the Version Log as a version number for the website.
This version number is then added as a property to the `WebSite` instance that is


example Protégé 3.1.1 (file:\D:\Sepulman\Home\University\thesis\Input\test\phase6\example.pprj, OWL Fi...
File Edit Project OWL Code OWL Code Window Tools Help
Individuals
Metadata
Version Log
Request Recovery
OWLClasses
Properties
Forms
Browse...
Set Website Version
Check URL
Build IM

Version Log	/thesis\Input\examplelogofversions.owl
Version Log Base	tp://wise.vub.ac.be/ontologies/log.owl#
URL to Check	
Snapshot URL	

Figure 9.6: After each implementation of the website, a version number is assigned to the Website.

a part of the IM currently being edited in Protégé.
This addition is done using the Java api provided by Protégé. The version number to add is determined from the Version Log using the Jena api and the WebSite instance to add the version number to is retrieved using the Change Definition Language query of Listing 7.10 in Section 7.3.2.2.3.

Figure 9.7 shows the website version number as a property of the `WebSite` instance named `web_0`. The version number here is 33, meaning that from the start of this example, 33 changes made to the IM have caused 33 new versions to be added to the Version Log.

Because the Version Log plugin is still monitoring the IM, the addition of the website version number is also recorded in the Version Log. The result of this addition is shown in Figure 9.8.
Note that the Transaction time of the version recording the addition corresponds to the website version number that is added to the IM. Because the Version Log maintains the preceding versions of the `web_0` `WebSite` instance, the website version numbers that are recorded by these versions can be retrieved and used by

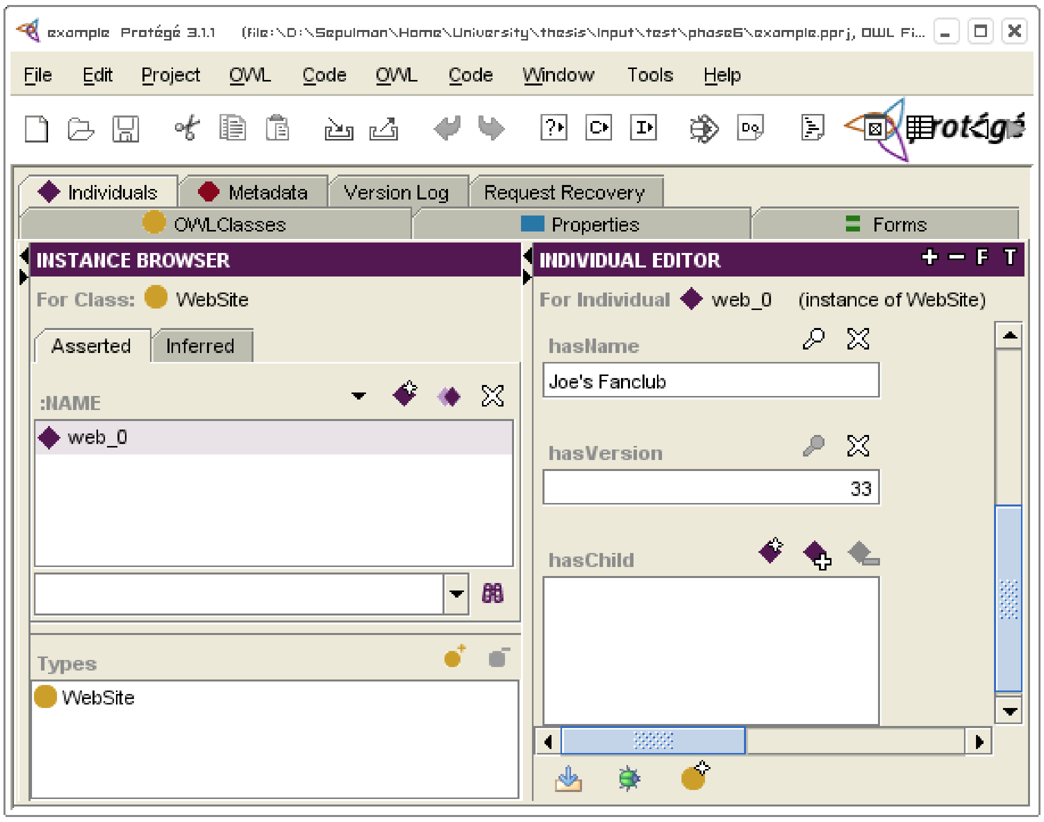

example Protégé 3.1.1 (file:\D:\Sepulman\Home\University\thesis\input\test\phase6\example.pprj, OWL Fi...
File Edit Project OWL Code OWL Code Window Tools Help
Individuals Metadata Version Log Request Recovery
OWLClasses Properties Forms
INSTANCE BROWSER
For Class: WebSite
Asserted Inferred
:NAME
web_0
Types
WebSite
INDIVIDUAL EDITOR
For Individual web_0 (instance of WebSite)
hasName
Joe's Fanclub
hasVersion
33
hasChild

Figure 9.7: The version number assigned to the version of the Website is present in the IM.

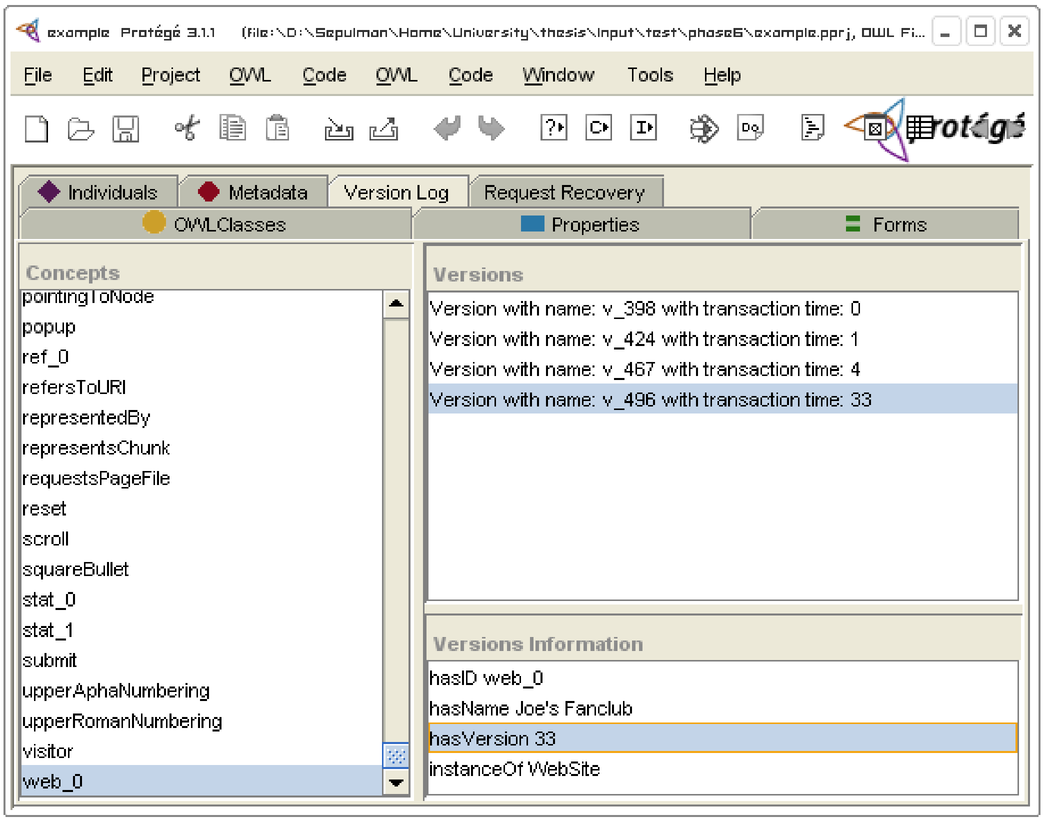
our approach.

9.2.2 Resolving a URL

The functionality described in this Section allows us to *resolve* an URL that contains a resource request for a resource of our website. By resolving an URL, we mean that we check if the resource request linked to that URL is valid, and if this is not the case, that we provide a new URL that corrects the potential error caused by the invalid resource request. One of the following cases can occur if the Request Recovery plugin is used to resolve an URL:

1. The resource request is valid: we notify the user of the plugin that this is the case;

2. The resource request is invalid and we detect a renaming change: the user is provided with an URL that contains the new resource request that reflects the results of the renaming change.



Concepts

pointingToNode
popup
ref_0
refersToURI
representedBy
representsChunk
requestsPageFile
reset
scroll
squareBullet
stat_0
stat_1
submit
upperAphaNumbering
upperRomanNumbering
visitor
web_0

Versions

Version with name: v_398 with transaction time: 0
Version with name: v_424 with transaction time: 1
Version with name: v_467 with transaction time: 4
Version with name: v_496 with transaction time: 33

Versions Information

hasID web_0
hasName Joe's Fanclub
hasVersion 33
instanceOf WebSite

Figure 9.8: The version number assigned to the version of the Website is versioned by the Version Log.

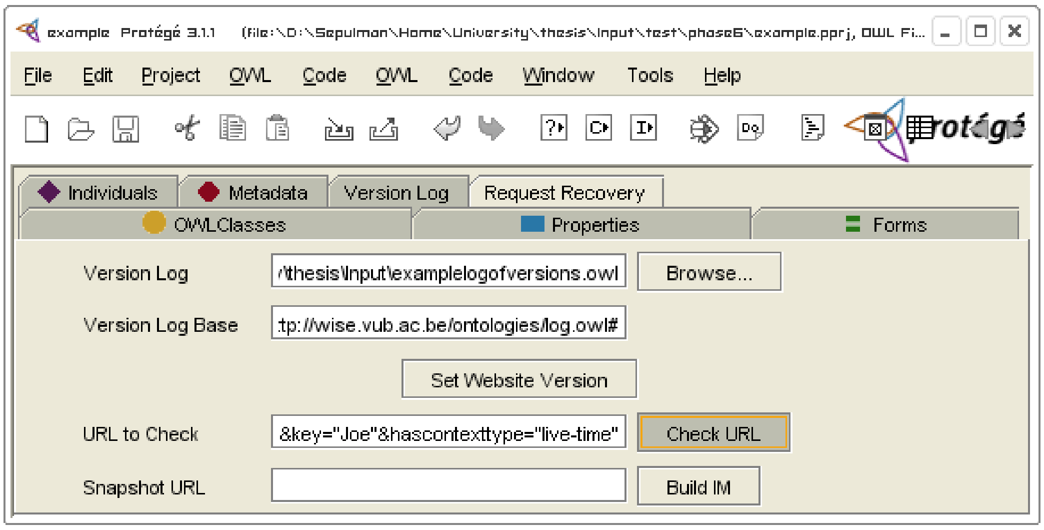
3. The resource request is invalid and we detect a removing change: the user is provided with an URL containing a snapshot resource request that requests the recovery of the IM to a version that is able to fulfill the original resource request

4. The resource request is invalid and we detect a removing change, but the we are not allowed to recover from this removing change: the user is provided with an URL containing the resource request for an error page indicating that the user may no longer view the resource that was requested.

5. The resource request is invalid and we detect that the resource request was never valid for any version of the website since its creation: the user is provided with an URL containing the resource request for an error page.

The Request Recovery plugin uses the Change Definition Language queries from the Listings of Section 7.3.2 to retrieve the necessary information from the Version Log. This information is then combined to resolve the URLs that the user of the plugin provides.

Figure 9.9 shows the URL from our example of Chapter 8 being used as input for the URL resolving component of the Request Recovery plugin.
The url that is used as input is the following:


example Protégé 3.1.1 (file:\D:\Sepulman\Home\University\thesis\input\test\phase6\example.pprj, OWL Fi...
File Edit Project OWL Code OWL Code Window Tools Help
Individuals Metadata Version Log Request Recovery
OWLClasses Properties Forms
Browse...
Set Website Version
Check URL
Build IM

Version Log	/thesis\input\examplelogofversions.owl
Version Log Base	tp://wise.vub.ac.be/ontologies/log.owl#
URL to Check	&key="Joe"&hascontexttype="live-time"
Snapshot URL	

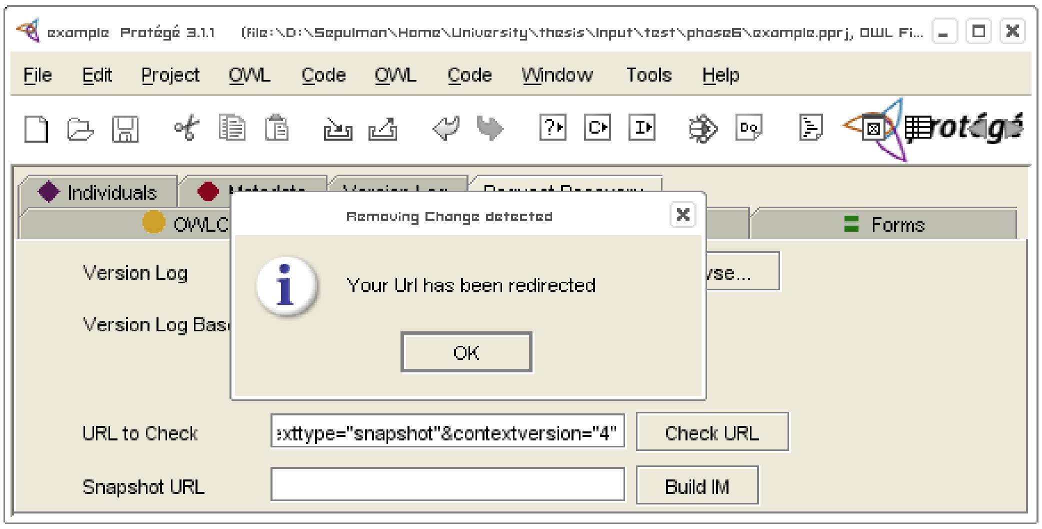
Figure 9.9: Checking an URL using the Request Recovery plugin.

```
http://www.example.com/gen?node="DescribeMember"&class
="Member"&key="Joe"&contexttype="live-time"
```

Because both a renaming and removing change have been performed on our example website, the URL that is shown to be returned in Figure 9.10 reflects both these changes. The URL that is returned by our plugin is:

```
http://www.example.com/gen?node="DescribeMember"&class
="Member"&key="TAFKAJ"&contexttype="snapshot"&context
version="4"
```

The renaming change is reflected in this URL by the change in key value from ”Joe” to ”TAFKAJ” and the removing change is reflected by the change in contexttype from ”live-time” to ”snapshot”.
The URL indicates that the IM should be recovered to the website version with version number ”4”. This version number is the number attached to the version that captures the link concept representing the resource request, just prior to its deletion.


example Protégé 3.1.1 (file:\D:\Sepulman\Home\University\thesis\Input\test\phase6\example.pprj, OWL Fi...
File Edit Project OWL Code OWL Code Window Tools Help
Individuals
OWLC
Forms
Removing Change detected
OK
Check URL
Build IM

Your Url has been redirected

Version Log

Version Log Bas

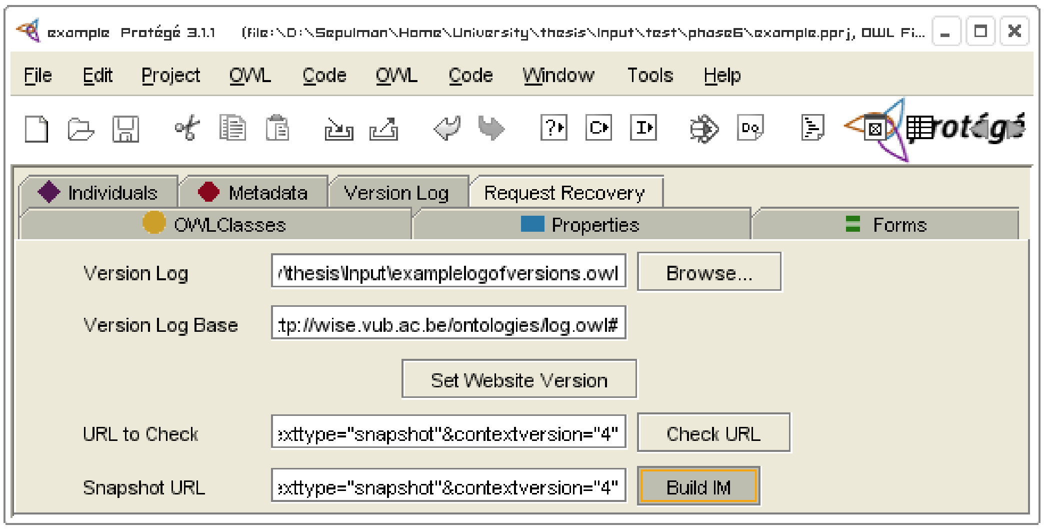
URL to Check: xttype="snapshot"&contextversion="4"

Snapshot URL

Figure 9.10: The result of checking an URL.

9.2.3 Recovering the IM

Snapshot type resource requests require that the IM used as a basis for the implementation of the website can be rolled back to a preceding version. This rolling back is achieved by transforming instances of the Version Log concepts (which are defined using the Version Ontology) back into the instances, properties and classes of the IM that they describe.
The Request Recovery plugin provides a simplified implementation of this functionality. In this proof of concept implementation, we have focused on providing the roll-back functionality for the instances of the Version Log describing the instances of the IM. Because the most of the classes and properties of the IM are fixed and described in the WSDM Ontology (with the exception of the classes and properties contained within the Object Chunks), we have limited the roll-back functionality for classes and properties to the extent that these classes and properties are needed to create instances.
Figure 9.11 shows the interface of the Request Recovery plugin that provides the roll-back functionality.
The user inputs a URL, in this case the URL that was returned in Section 9.2.2. The resource request for this URL has as context version number the number 4. We then retrieve all the concepts from the Version Log that have as version number the number 4. We do this by using the Change Definition Language query from Listing 7.5 in Section 7.3.2.1.3. Next, we use Jena to retrieve these concepts and their properties from the Version Log and to create a new OWL file that will contain the rolled back IM. We fill this OWL file by going through all the retrieved

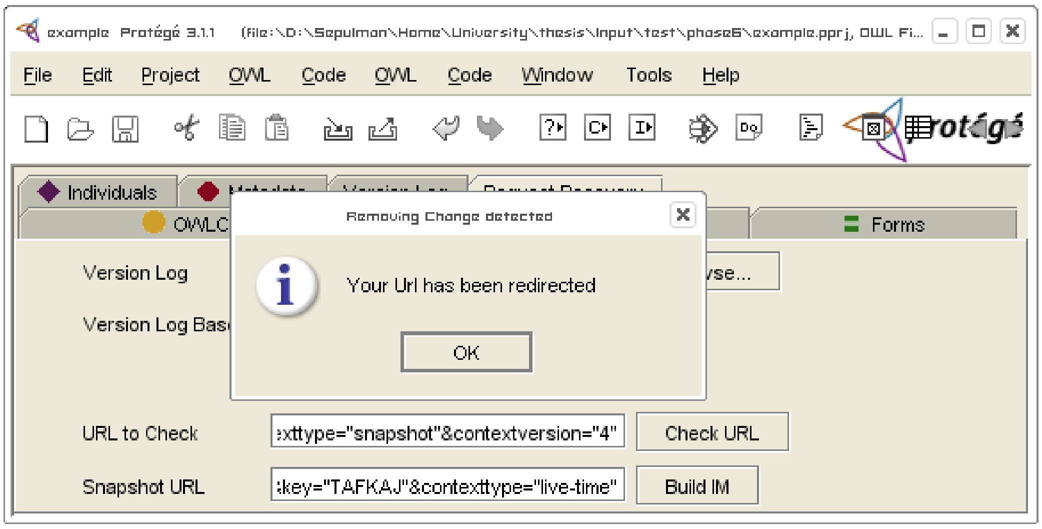

example Protégé 3.1.1 (file:\D:\Sepulman\Home\University\thesis\Input\test\phase6\example.pprj, OWL Fi...
File Edit Project OWL Code OWL Code Window Tools Help
Individuals Metadata Version Log Request Recovery
OWLClasses Properties Forms
Browse...
Set Website Version
Check URL
Build IM

Version Log: /thesis\Input\examplelogofversions.owl
Version Log Base: tp://wise.vub.ac.be/ontologies/log.owl#
URL to Check: xttype="snapshot"&contextversion="4"
Snapshot URL: xttype="snapshot"&contextversion="4"

Figure 9.11: Rebuilding the IM using the Request Recovery plugin.

concepts and transforming these concepts and their associated classes, properties and instances as we encounter them.

Now that we have the IM, we need to adapt the resource request to this rolled back IM. We do this by transforming the snapshot type resource request into a live-time resource request. For our example, this means that we transform:
`http://www.example.com/gen?node="DescribeMember"&class="Member"&key="TAFKAJ"&contexttype="snapshot"&contextversion="4"`
into:
`http://www.example.com/gen?node="DescribeMember"&class="Member"&key="TAFKAJ"&contexttype="live-time"`
and return this url to the user, as is show in Figure 9.12



Removing Change detected

Your Url has been redirected

URL to Check: :xttype="snapshot"&contextversion="4"

Snapshot URL: :key="TAFKAJ"&contexttype="live-time"

Figure 9.12: Returning the transformed URL

Chapter 10

Conclusion

In this thesis, we have presented an approach that makes use of the meta-information provided by ontology-based Website Design Methods to correct broken-link type errors for the websites created using these ontology-based website design methods.
This meta-information is contained in the models of the website design method. These models describe the website that is being created and they are in turn described using an ontology language. These models are sufficiently detailed to transform them into a complete website implementation.

In this thesis we have used the WSDM ontology-based Website Design Method, that has been developed at the WISE Laboratory. The models of this Website Design Method are described by the WSDM Ontology described in turn in the OWL Ontology Language. The models of the WSDM Method are basicly saved on disk as files containing OWL statements.
Our approach uses another idea developed at the WISE Laboratory, namely a Version Log, this Version Log is another OWL file that logs (records) all the changes made to an ontology. The ontology in this case, are the OWL files that contain the models that describe the website.
Because the Version Log records the changes made to the models of the website (and thus the website itself) in sufficient detail, we can use the information in this log to rebuild previous versions of the models. Since these models can be transformed into a complete website, previous versions of the models can be transformed into a complete previous version of the website.
Because broken link type errors are caused by changes in the website (and thus changes in the models), we can use the information in the Version Log to detect these changes and rebuild a version of the website pre-dating these changes, thereby correcting the broken link error.

The advantages of the approach described in this thesis are the following:

- Our approach can be implemented as a fully automated approach. The Multimedia datasource versioning, the website versioning and the link extension versioning algorithms described in Section 7.3.2.2 can all be set to automatically trigger at some point during the implementation phase. The Change Handling algorithms from Section 7.3.2.1 are all designed to produce different results, depending on the type of resource request that is used as input.

- For a website designed by using the WSDM method, no extra input is required from the website engineer to enable our error-correction approach. We can simply import the existing IM into our method, generate a Version Log for that IM and continue with the rest of the implementation phase;

- The usage of ontologies as the source of information for our error-correction approach, allows us to import models from other ontology-based web design methods and transform these models into WSDM models, that can be used by our error-correction approach.

- Extra extensions (e.g.: The Temporal Context Type from Section 7.3.1.1 and the Deletion Extensions from Section7.3.1.2) are provided to enable the website engineer to better express the temporal relations for link and resource removal, but the website engineers are not obligated to use them.

We have shown in this thesis that it is possible to use the meta-information of an ontology-based website design method to provide an effective error-correction approach for the website that has been generated by using this ontology-based website design method. The error-correction approach presented in this thesis is yet another advantage of the usage of structured, ontology based website design methods.

We finalize by describing some possible further extensions of the work presented in this thesis;

- Most obviously it would be interesting to integrate our implementation with an implementation of the WSDM transformational pipeline. We would then be able to check the extent to which our approach meets the expectations of the users and the website engineers.

- Another interesting extension would be, adding the possibility for conditional temporal contexts. These conditional temporal contexts would appear under the form of Change Definition Language queries, of which the result of the execution determines which version of a resource is returned for the resource request.

Chapter 11

Acknowledgements

First of alI, I would like to thank my promoter, Prof. Dr. Olga De Troyer for giving me the opportunity to create my thesis and serve my internship at the WISE laboratory.
Special thanks go to Peter Plessers, for supporting me during my internship and the creation of this thesis. His helpful remarks and proofreading have most certainly improved the quality of this thesis.
Finally, I would like to thank my family for supporting me both morally and financially for all these years.

Bibliography

[1] Berners-Lee, T., Cailliau, R., 1990, ”World Wide Web: Proposal for a HyperText Project” - CERN European Laboratory for Particle Physics, Geneva CH, 12 November 1990.

[2] Bush, V., 1945, ”As we may think”. The Atlantic Monthly, Vol. 176, Nr. 1,Ed. Weeks E. A.,pp. 101-108, ISSN 1072-7825, July 1945

[3] Berners-Lee, T., 1994, ”Universal Resource Identifiers in WWW”, Internet Requests For Comments (RFC) 1630

[4] Rimmer, J., I. Wakeman, L. Sheeran, and M.A. Sasse, 2000, ”Messages from a Tangled Web.” In Proceedings of OzCHI, Eds . Paris C., Ozkan N., Howard S., Lu S., ISBN 0-643-06633-0, Sydney, Australia, December 2000

[5] Kelley, B., ”Approaches to the preservation of Websites”, 2002, Online Information 2002 conference, ”Archiving the web: tackling digital preservation” session, Olympia, London, 3 December 2002.

[6] Fielding, et al., ”Hypertext Transfer Protocol – HTTP/1.1”, 1999, Internet Requests For Comments (RFC) 2616, 1999

[7] Koehler, W., 2004, ”A longitudinal study of Web pages continued: a consideration of document persistence.”, In Information Research, (paper 174), Vol. 9 No. 2, ISSN 1368-1613, January 2004

[8] Kelly, B., ”Guidelines For URI Naming Policies”, 2002, Ariadne, Issue 31, Ed. Hunter P., Pub UKoln, ISSN 1361-3200, March 2002,

[9] Chen, C.C., Chung, Y.C., Chien, C.C., Lee, C. 2004. ”Similarity retrieval of web documents, considering both text and style.”, Lecture Notes In Computer Science, Iss. 3007, pp. 620-629, Publ. Springer-Verlag, Germany, ISSN 0302-9743, 2004

[10] De Troyer, O., 1998, ”Designing Well-Structured Web Sites: Lessons to be Learned from Database Schema Methodology”, In Proceedings of the ER ’98 Conference, Lecture Notes in Computer Science 1507, pp. 51 - 64, Eds. Ling T.W., Ram S., Lee M.L., Publ. Springer-Verlag, ISBN 3-540-65189-6, Singapore, November 1998

[11] Gruber, T., ”A Translation Approach to Portable Ontology Specifications.”, 1993, In Knowledge Acquisition, Vol. 5, No. 2, pp. 199-220, Publ. Academic Press, ISSN 0001-2998, April 1993

[12] Gruber, T., 1993, ”Toward principles for the design of ontologies used for knowledge sharing.”, International Journal of Human-Computer Studies, special issue on Formal Ontology in Conceptual Analysis and Knowledge Representation, Eds. Guarino N., Poli R., Publ. Academic Press, ISSN 1071-5819, technical report, KSL-93-04, Knowledge Systems Laboratory, Stanford University, 1993

[13] Plessers, P., De Troyer, O., 2005 ”Ontology Change Detection using a Version Log”, In Proceedings of the 4th International Semantic Web Conference, pp. 578-592, Eds. Yolanda Gil, Enrico Motta, V.Richard Benjamins, Mark A. Musen, Publ. Springer-Verlag, ISBN 978-3-540-29754-3, Galway, Ireland 2005

[14] Plessers, P., De Troyer, O., 2004, ”Annotation for the Semantic Web during Website Development”, In Proceedings of the ICWE 2004 Conference, Lecture Notes in Computer Science 3140, pp. 349-353, Eds. Nora Koch, Piero Fraternali, and Martin Wirsing, ISBN 3-540-22511-0, Munich, Germany 2004

[15] Plessers, P., De Troyer, O., 2004, ”Web Design for the Semantic Web”, In Proceedings of the WWW2004 Workshop on Application Design, Development and Implementation Issues in the Semantic Web, CEUR Workshop Proceedings, Vol 105 Web, WWW2004 Workshop, Eds. Christoph Bussler, Stefan Decker, Daniel Schwabe, Oscar Pastor, ISBN 1613-0073, New York, USA 2004

[16] Vdovjak, R., Frasincar, F., Houben, G. J., Barna, P., 2003, ”Engineering Semantic Web Information Systems in Hera”, Journal Of Web Engineering, Vol. 2, No.1&2, pp. 003-026, Publ. Rinton Press, ISSN 1540-9589, Princeton, New Jersey, September 2003

[17] Y. Lei, E. Motta, and J. Domingue, 2004, ”Modeling Data-Intensive Web Sites with OntoWeaver”, In Proceedings of the International Workshop on

Web Information Systems Modeling (WISM 2004), Eds. Frasincar F., Vdovjak R., Houben G-J.Barna P., Publ. Springer-Verlag, Riga, Latvia, June 2004

[18] Berners-Lee, T., Hendler and J., Lassila, O. 2001.”The Semantic Web.” In Scientific American, pp. 34-43, Publ Scientific American, Inc., ISSN 0036-8733, New York, USA,May 2001

[19] Koivunen, M.R., Miller, E., 2001, ”W3C Semantic Web Activity”, Semantic Web Kick-Off in Finland - Vision, Technologies, Research, and Applications, Ed.Hyvnen E., Publ.HIIT Publications, ISBN 1458-951-22-6019-0, Helsinki, Finland, Nov 2, 2001

[20] Shwabe, D., Rossi, G., 1998, ”An object oriented approach to web-based applications design”, Theory and Practice of Object Systems, Vol. 4, Nr. 4, pp. 207-225, Eds Lieberherr K., Zicari R., Publ. John Wiley & Sons, ISSN 1074-3227, 1998.

[21] Garrigs, I., Gmez, J., Cachero, C., 2003, ”Modeling Dynamic Personalization in Web Applications”, In Proceedings of ICWE 2003, Lecture Notes in Computer Science, Vol. 2722, pp. 472 - 475, Eds Lovelle J.M.C, Martín González Rodríguez B., Aguilar L.J., Labra Gayo J.E. del Puerto Paule Ruíz M., Oviedo, Publ. Springer-Verlag, ISBN 3-540-40522-4, Spain, July 2003

[22] R. Hennicker and N. Koch., 2000, ”A UML-based Methodology for Hypermedia Design”, Proc. of UML 2000 Conference, Lecture Notes in Computer Science, Vol 1939, Eds Evans A., Stuart A., Selic B., Publ. Springer Berlin / Heidelberg, ISSN 0302-9743, York, England, October 2000

[23] Isakowitz, T., Stohr, E.A., Balasubramaninan, P., 1995, ”RMM: A Methodology for Structured Hypermedia Design”, Communications of the ACM, Volume 38, Issue 8, pp. 34 - 44 ,Publ. ACM Press, ISSN 0001-0782, New York, NY, USA, August 1995

[24] Chen, P.P. -S., 1976, ”The entity-relationship model: toward a unified view of data”, ACM Transactions on Dastabase Systems, Volume 1, Issue 1, pp. 9-36, Publ. ACM Press, ISSN 0362-5915, Framingham, MA, March 1976

[25] Fowler, Martin., 2003, ”UML Distilled: A Brief Guide to the Standard Object Modeling Language”, 3rd ed., Publ. Addison-Wesley. ISBN 0321193687, 2003

[26] De Troyer, O., Leune, C, 1998, ”WSDM: A User-Centered Design Method for Web Sites”, Proceedings of the seventh international conference on

World Wide Web 7, Pages: 85 - 94, Eds. Enslow P.H. Jr., Ellis A., Publ. Elsevier Science Publishers B. V., ISSN 0169-7552, Brisbane, Australia, 1998

[27] Jin, Y., Decker, S., Wiederhold, G., 2001, ”OntoWebber: Model-Driven Ontology-Based Web Site Management”, Proceedings of SWWS’01, The first Semantic Web Working Symposium, Eds. Cruz I.F., Decker S., Euzenat B., McGuinness D.L., Publ. Stanford University, California, USA, July 29 - Aug 1, 2001.

[28] Klapsing, R., G Neumann, G., 2000, ”Applying the Resource Description Framework to Web Engineering”, Lecture Notes In Computer Science: Vol. 1875, Proceedings of the First International Conference on Electronic Commerce and Web Technologies, Pages: 229 - 238, Eds. Kittler J., Roli F., Publ.Springer Berlin / Heidelberg, ISBN 3-540-67981-2, Cagliari, Italy, June 2000

[29] Houben, G. J. , Frasincar, F., Barna, P., Vdovjak, R., 2004, ”Modeling User Input and Hypermedia Dynamics in Hera”, In ICWE2004, International Conference on Web Engineering, LNCS 3140, pp. 60-73, Eds. Koch N., Fraternali P., Wirsing M., Publ. Springer, ISBN 3-540-22511-0, Munchen, Germany, pp. 26-30 July 2004

[30] Fiala, Z., Frasincar, F., Hinz, M., Houben, G.J., Barna, P., Meissner, K., 2004, ”Engineering the Presentation Layer of Adaptable Web Information Systems”, In ICWE2004, International Conference on Web Engineering, LNCS 3140, p. 459-472, Eds. Koch N., Fraternali P., Wirsing M., Publ. Springer, ISBN 3-540-22511-0, Munchen, Germany, 26-30 July 2004

[31] Fiala, Z., Hinz, M., Meissner, K., Wehner, F.: A , 2003, ”Component-based approach for adaptive dynamic web documents”, Journal of Web Engineering, Vol.2 No.1&2, pp. 058-073, Publ. Rinton Press, ISSN 1540-9589, Princeton, New Jersey, September, 2003

[32] Lima F., Schwabe, D., 2003 ”Application Modeling for the Semantic Web”, LA-WEB 2003 - First Latin American Web Conference, Santiago, Chile, 2003 - IEEE-CS Press.

[33] Schwabe, D., Szundy, G., de Moura, S. S, Lima, F., 2004, ”Design and Implementation of Semantic Web Applications”, Proc. of the Workshop on Application Design, Development and Implementation Issues in the Semantic Web (WWW 2004), CEUR Workshop Proceedings, Vol. 105, Eds. Christoph Bussler and Decker S., Schwabe D.,Pastor O., ISSN 1613-0073, New York, NY, USA, May 18, 2004.

[34] De Troyer, O., Casteleyn, S., 2003, ”Modeling Complex Processes for Web Applications using WSDM”, In Proceedings of the Third International Workshop on Web-Oriented Software Technologies (held in conjunction with ICWE2003), IWWOST2003 (also on `http://www.dsic.upv.es/~west/iwwost03/articles.htm`), Eds. Daniel Schwabe, Oscar Pastor, Gustavo Rossi, Luis Olsina, Oviedo, Oviedo, Asurias, Spain July 15, 2003

[35] Paterno, Mancini, Meniconi, 1997. ”ConcurTaskTrees: a Diagrammatic Notation for Specifying Task Models”, In Proceedings of INTERACT 97, IFIP TC13, International Conference on Human-Computer Interaction, Vol 96, pp. 362-366, Eds. Howard S., Hammond J., Lindgaard G., Publ. Chapman & Hall, ISBN 0-412-80950-8, Sydney, Australia, 14th-18th July 1997

[36] Halpin, T., 2001. ”Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design”, 1st Edition. Publ. Morgan Kaufmann Publishers, ISBN 1558606726

[37] Lei, Y. , Motta, E., Domingue, J., ”OntoWeaver - an ontology based approach to web site design and development”, see `http://kmi.open.ac.uk/projects/akt/ontoweaver/`

[38] Lei, Y. , Motta, E., Domingue, J., 2003, ”Design of Customized Web Applications with OntoWeaver (2003).”, In proceedings of the International Conference on Knowledge Capture, pp 54-61, Eds. Gennari J.,Porter B.,Gil Y., Publ. ACM Press, ISBN 1-58113-583-1, Sanibel Island, FL, USA, October 23 - 25, 2003

[39] Lei, Y. , Motta, E., Domingue, J., 2002, ”An Ontology-Driven Approach to Web Site Generation and Maintenance.”, The 13th International Conference on Knowledge Engineering and Management (EKAW 2002), Lecture Notes in Computer Science 2473, pp. 219-234, Ed. Gomez-Perez, A., Publ. Springer Verlag, ISBN 3-540-44268-5. Sigenza, Spain 1-4 October 2002

[40] De Troyer, O., Casteleyn, S., Plessers, P., 2005, ”WSDM+: Exploiting Semantic Web Technology during Web Site Design”, Technical Report, 2005.

[41] Casteleyn, S., 2005, ”Designer Specified Self Re-organizing Websites”, Phd thesis, Vrije Universiteit Brussel, 2005

[42] Stojanovic, L., 2004, ”Methods and Tools for Ontology Evolution”, Phd Thesis, University of Karlsruhe, 2004

[43] Klein, M., 2004, ”Change Management for Distributed Ontologies”, Phd Thesis, Vrije Universiteit Amsterdam, ISBN 90-9018400-7, August 2004