Specification of Application Logic in Web Information Systems

Peter Barna
Specification of Application Logic in Web Information Systems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus, prof.dr.ir. C.J. van Duijn,
voor een commissie aangewezen door het College voor Promoties
in het openbaar te verdedigen
op donderdag 30 oktober 2007 om 16.00 uur

door

Peter Barna

geboren te Krompachy, Slowakije
Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. P.M.E. De Bra
en
prof.dr.ir. G.J.P.M. Houben
Contents

Preface ix

1 Introduction 1
   1.1 Motivation .................................................. 1
   1.2 Aim of this Dissertation .................................. 2
      1.2.1 Basic Application Model ............................ 3
      1.2.2 Advanced Application Model ....................... 3
      1.2.3 Design Support ....................................... 3
      1.2.4 Software Tools ...................................... 4
   1.3 Research Topics ........................................... 4
   1.4 Dissertation Outline ..................................... 5

2 Methods for Web Information System Design 7
   2.1 Methods for WIS Design .................................. 7
      2.1.1 Relationship Management Model (RMM) ............. 7
      2.1.2 Web Modeling Language (WebML) .................... 8
      2.1.3 Object-Oriented Hypermedia Design Model (OOHDM) 9
      2.1.4 UML-Based Web Engineering Method (UWE) .......... 10
      2.1.5 Object-Oriented Hypermedia Model (OO-H) .......... 11
      2.1.6 Web Site Design Method (WSDM) .................... 12
      2.1.7 The Hera Method .................................. 13
   2.2 Hera Extension ............................................. 14
      2.2.1 Extended Hera Architecture ...................... 14
      2.2.2 Extended Hera Models ................................ 14
   2.3 Conclusion .................................................. 16

3 Domain and Basic Application Modeling 17
   3.1 Domain Modeling ............................................. 17
   3.2 Basic Application Modeling ............................... 23
      3.2.1 Data Accessors ....................................... 24
      3.2.2 Navigation Units ................................... 26
      3.2.3 Application Model .................................. 32
   3.3 Conclusion .................................................. 38
# Advanced Application Modeling

4.1 Overview of the Advanced AM ............................................. 39
4.2 Extending the Basic AM Structure ....................................... 41
4.3 Navigation Model in the Advanced AM ................................. 43
   4.3.1 Motivation .......................................................... 43
   4.3.2 Forms .............................................................. 43
   4.3.3 Inclusion Conditions .............................................. 45
   4.3.4 Conditional Navigation ......................................... 47
   4.3.5 Inheritance in NM .................................................. 49
4.4 Content Manipulation Model .............................................. 50
   4.4.1 Motivation .......................................................... 51
   4.4.2 Events .............................................................. 51
   4.4.3 Conditions .......................................................... 56
   4.4.4 Actions .............................................................. 56
   4.4.5 CMM Vocabulary ................................................... 62
4.5 The Extended AM .......................................................... 63
   4.5.1 AM Re-Definition .................................................. 63
   4.5.2 AM Instantiation ................................................... 64
   4.5.3 Example AM ......................................................... 65
4.6 Conclusion ................................................................. 70

# Modeling Collaboration in Web Information Systems

5.1 Overview of Process Modeling Techniques .............................. 72
5.2 Motivation ................................................................. 73
5.3 Task Model ............................................................... 75
5.4 Collaboration Model ..................................................... 76
   5.4.1 Activity Diagrams Summary ..................................... 78
   5.4.2 Expressing Collaborative Systems Using ECA Rules ........ 82
5.5 Specification of Activities .............................................. 85
   5.5.1 Information Observation ......................................... 85
   5.5.2 Information Entry .................................................. 86
   5.5.3 Information Updates .............................................. 87
   5.5.4 Defining Events .................................................... 88
   5.5.5 Expressing ECA Rules ............................................ 89
5.6 Generation of Application Model ....................................... 91
   5.6.1 Generation of the Content Manipulation Model .............. 92
   5.6.2 Generation of the Navigation Model ......................... 101
   5.6.3 Generation Procedure ............................................ 105
5.7 Conclusion ............................................................... 106
CONTENTS

6 Application Templates 109
  6.1 Reuse in WIS design ........................................... 109
  6.2 Application Templates Overview .................................. 110
    6.2.1 AT Interfaces ........................................... 111
    6.2.2 AT Deployment .......................................... 112
  6.3 Application Templates in Hera ................................... 113
    6.3.1 Target Domain Mapping .................................. 113
    6.3.2 Publication Example ...................................... 115
    6.3.3 Mapping Conflicts ...................................... 121
    6.3.4 Guided Tour Example .................................... 122
  6.4 Applying Mappings ............................................ 127
    6.4.1 Mapping Vocabulary .................................... 127
    6.4.2 Mapping Procedure ..................................... 128
  6.5 Conclusion .................................................... 128

7 Software Tools 131
  7.1 Hera Presentation Generator ................................... 131
    7.1.1 HPG Interfaces and Configuration .......................... 132
    7.1.2 Requests Processing Procedure ............................ 133
    7.1.3 Decoding Requests ...................................... 134
    7.1.4 Generating Events and CMM Rule Triggering ............... 135
    7.1.5 Generating Pages ........................................ 136
    7.1.6 Executing Retrieval and Content Manipulation Queries .. 137
    7.1.7 Timers and External Communication ....................... 144
    7.1.8 Implementation Remarks .................................. 145
  7.2 Support for the Construction of Design Models ................. 146
  7.3 Support for Deployment of Application Templates .............. 147
  7.4 Conclusion .................................................... 147

8 Concluding Remarks 157
  8.1 Conclusions .................................................. 157
  8.2 Future Work .................................................. 159

Bibliography 161

Index 167

Summary 171

Samenvatting 173

Curriculum Vitae 175

SIKS Dissertatiereeks 177
Acknowledgements

I would like to thank my supervisors professor Geert-Jan Houben and professor Paul De Bra for their great support during my research. Professor Geert-Jan Houben as my daily supervisor was always able to find time for discussions and brought numerous inspiring ideas. His great scientific overview and organization skills helped me to solve a number of research and administrative issues. Professor Paul De Bra was ready to help at any moment too and it was a great pleasure and opportunity for me to discuss different topics with him. I had a great time to cooperate with both my supervisors.

During my studies I cooperated with many people that influenced my research. Flavius Frasincar and Richard Vdovjak were not only my colleagues with whom I was in daily contact but also good friends. Our countless discussions had a great influence on my work and resulted in a number of publications we wrote together. I want to thank my colleagues Kees van der Sluijs, Pieter Bellekens, Ad Aerts and Philippe Thiran for their cooperation and the friendly working environment they created. Master students Bas Rutten, Bert Okkerse, Martin Schuijers and Laurence Mees made a great effort implementing and extending tools for the Hera project and I want to thank them. During my PhD study I had the pleasure to discuss and work with my fellow PhD students from different universities Roberto De Virgilio, Irene Garrigos, Zoltan Fiala, Sven Casteleyn and Peter Plessers. I am grateful for their insight, informal talks, and for their friendship. The secretary of my research group Riet van Buul greatly helped me to correct the text of this dissertation and I want to thank her for her participation. All my friends and present colleagues deserve special thanks for their support and motivation.

Most of all I would like to thank my parents without whom none of this would have been possible.
Chapter 1

Introduction

1.1 Motivation

The World Wide Web (the Web) has grown into a phenomenon having a huge impact on society. The Web has influenced the daily life of millions of people and their customs. Data available on the Web has undoubtedly become the largest information source and the Web itself has become the largest communication platform. Its success came first with early Internet technology and later with fast adoption of simple and easy to use network hypertext protocols such as HTTP and HTML. Together with the first available Web browsers a myriad of handcrafted hypertext documents were published online.

The success and rapid growth of the Web with a swiftly growing number of users has quickly outrun tools for weaving the Web - technologies and methods for designing Web applications. Techniques used for building hypertext documents and first real Web information systems (WIS) [Isakowitz et al., 1998] soon proved to be insufficient to cope with the large potential of the Web. One of the reasons was undoubtedly the fact that even creators of early Web protocols and software together with authors of early web documents could not foresee the impact and popularity the Web would gain.

One of the important milestones in the history of the Web is the understanding that due to the large and growing mass of information and the high demand on labor effort involved, handcrafting is not a conceivable way for building Web sites. In order to provide up-to-date and well-organized information more intelligent techniques are needed than storing information in static hyperdocuments. Such techniques would allow retrieving (and restructuring) data content on demand from (potentially dynamically changing) data sources. Data-intensive WIS are typically built on top of database warehouses and thus constitute what is called the “deep Web”. Another important fact is that WIS tend to evolve from information providing facilities to full-fledged systems providing complex services such as e-commerce and institutional portals, online shops, libraries, etc., where the process of user-system collaboration gains large importance. The development of such complex WIS obviously requires a disciplined and well organized approach.
Using well-established (traditional) software engineering methods for building WIS has been a WIS design starting point and it still is the most common approach. However, while some aspects of WIS are similar to traditional information systems and traditional methods are perfectly suited here (domain/conceptual modeling), some aspects of WIS are unique. The most obvious characteristic is the Web interface that to a large extent influences the way users interact with WIS. Another remarkable feature of most WIS is a huge user audience. Typically, thanks to Web availability many WIS are used by a large number of people with varying skills, knowledge, cultural backgrounds and interests, which is not a typical case for traditional IS. Advances in wireless technology and home electronics facilitate exploitation of Web applications using different platforms. All the mentioned aspects should be considered when designing WIS. Since these aspects are far from trivial the development of software for the Web asks for dedicated methods. A number of methods aimed at WIS design has been developed and they are summarized in the next Chapter. These methods describe a WIS by a set of models for different facets of the system. The division of different system aspects into separate model specifications contributes to clarity and maintainability of the design.

1.2 Aim of this Dissertation

This dissertation presents an extension of the Hera WIS design method [Frasincar et al., 2001, 2006]. The Hera method was developed to support the design of web presentations of semantically annotated data. The method evolved to a full-fledged software design method for WIS. The extension discussed in this dissertation includes improvement of the application model by a number of application modeling primitives and adding design support models. The application modeling has been extended in the following steps:

- **Step 1**: Summarizing existing Hera AM (referred to as basic AM) in terms of definition of its model vocabulary and semantics.

- **Step 2**: Adding definitions for a number of modeling terms for capturing user input and data content manipulations (advanced AM); extending its semantical specification.

- **Step 3**: Adding a lightweight process model that supports building AM for WIS requiring more extensive collaboration of multiple users.

- **Step 4**: Adding an AM template technique that allows reusing the same AM or its part for different domain models.

- **Step 5**: Extending (and re-implementing) existing software prototypes to validate (selected) modeling concepts from previous steps.

While the Hera AM is directly extended in Step 2, Steps 3 and 4 aim at facilitation and simplification of the design process. Step 5 covers the validation of proposed concepts using software prototype tools.
1.2 Aim of this Dissertation

Note that the model extensions preserve the Semantic Web support provided by the original Hera models. The described goals are explained in the following sections and are projected to the research topics in Section 1.3.

1.2.1 Basic Application Model

The basic AM (Step 1) facilitates a format-independent specification of hypertext presentations. For this purpose it includes means for the specification of the content presented within visited navigation nodes (pages) and means for the specification of hyperlinks connecting the navigation nodes. These original Hera AM modeling elements (navigation nodes and hyperlinks) are extended with powerful data retrieval facilities. The modeling elements are formally defined and expressed as the RDFS application vocabulary (as in the original Hera AM).

1.2.2 Advanced Application Model

The difference between the basic and advanced AM is the ability of the advanced AM (Step 2) to express data content manipulations and user information entry. An important aspect taken into account when proposing the basic AM extension is the way how the basic and advanced AM are deployed and instantiated (populated with data) into hypertext presentations. Such presentations can be built either only once prior to their deployment, or (preferably) they can be built dynamically on demand at runtime. The advantage of the second approach is that the pages are always populated with fresh data. Because the data content can be changed within a browsing session it is obvious that the extensions contained in the advanced AM rely on the runtime instantiation of the hypertext presentation.

1.2.3 Design Support

The design support techniques described here fall into two different categories. The first technique helps to model Web applications that involve extensive communication (and collaboration) between users that happens beyond the scope of single user sessions, and the second technique helps to save the design effort by reusing AM for similar domains.

Design of Collaborative Web Applications

Modern Web applications often require communication between multiple users. Building AM for such applications requires considering issues like synchronization of tasks of different users and partitioning the navigation structure into chunks that can be visited/used without eventual blocking caused by waiting for communications from cooperating users. A model explicitly capturing the task synchronization based on UML activity diagrams together with automated procedure of its transformation to AM proposed in this work facilitates the design of such applications (Step 3).
AM Reuse

Software design is generally a laborious and costly process. Repeated usage and deployment of software design deliverables at different abstraction levels helps to lower the design effort. The approach described here introduces so called AM templates that can be reused for classes of similar domains with structurally different domain models (Step 4). The templates are deployed using mappings of virtual template domain concepts to real domain concepts.

1.2.4 Software Tools

Software prototyping helps to evaluate usability of investigated methods. It allows experimenting with a researched method and uncovers its positive and negative aspects. Software prototypes moreover help to detect and correct possible imprecisions or inconsistencies in proposed modeling concepts that become clear during the process of their design and implementation. Tools related to the discussed topics (Step 5) include tools for the specification and deployment of models (model builders, AM template mapper), and the software application generating hypertext presentations based on the proposed models (Hera Presentation Generator).

1.3 Research Topics

The dissertation goal described in the previous section is transformed into the following research topics that predetermine the structure of the further text:

**Research Topic 1: Application logic and its specification in the context of WIS**

In order to address the goals described in Section 1.2 we first need to clarify our understanding of the term application logic. Next, a summary of techniques used by different WIS design methods for expressing the application logic helps to understand the proposed Hera extension. In addition, the proposed Hera extension is briefly sketched and summarized.

**Research Topic 2: Modeling web interfaces built on top of data domains**

To address Step 1 in Section 1.2 the existing Hera application model is discussed and formalized. The basic Hera application model is defined by extending the existing modeling elements. This model includes easy to use and flexible data retrieval and presentation facilities allowing runtime construction of hypertext presentations.

**Research Topic 3: Fully-fledged WIS application modeling**

Most WIS require storing and maintaining the system state both to volatile and persistent data repositories. Next to the specification of the navigation structure (presentation logic) the content manipulations (business logic) should be specified in suitable models. Keeping the specification of business and presentation logic separate helps to maintain and extend
Research Topic 4: Modeling of WIS requiring extensive user collaboration
Recent WIS are often service-oriented rather than information-oriented. The services often involve participation of multiple types of users that needs to be synchronized. When the design of such systems is driven primarily by the navigation specification it usually fails because the navigation specification does not capture the synchronization of different parts of a workflow performed by different participants. Relationships between such task-oriented system specification and application modeling are clarified and practical conclusions are drawn. This research topic reflects the first sub-goal of Step 3 in Section 1.2.

Research Topic 5: Reuse of application models in the context of the Hera method
The technique addressed by Step 4 in Section 1.2 represents a reduction of design effort by reusing application specifications. Reusing design artifacts saves development costs and makes design more transparent. Sets of AM templates can be prepared for sets of typical problems (and domains). Such a template AM can be applied to concrete domains by using proper mappings. The effectiveness of this approach however depends on proximity of pre-designed template domain models and concrete domain models.

Research Topic 6: Validation of proposed models
Building prototype software tools as stated in Step 5 in Section 1.2 provides an important input for evaluation and validation of the discussed modeling concepts and methods. Using the tools allows third parties (e.g. students) to provide useful feedback. The main ideas are mapped to concrete modeling elements processed by concrete software modules.

1.4 Dissertation Outline
This dissertation consists of eight chapters and every chapter starts with a brief summary and motivation. Most of the related work is explained in Chapter 2 but some additional aspects of the related work are pointed out where it is needed.

Chapter 2 focuses on Research Topic 1. This chapter discusses existing methods supporting the design of WIS and investigates how the functionality (application logic) of WIS is defined and which models to use within these methods. The modeling techniques used by different methods are compared. Next the original Hera AM model is explained and the refined set of Hera models is summarized. The Hera framework is described and the position of refined Hera models is explained within this framework.

Chapter 3 covering Research Topic 2 redefines the basic Hera AM based on the model described in [Frasincar et al., 2002]. This model covers the structured specification of navigation structure with advanced data retrieval facilities. The application vocabulary is expressed using RDFS.
Chapter 4 addresses Research Topic 3 by extending the navigation specification in the basic AM with the content management model. This extension allows modeling of the business logic of full-fledged WIS. The advanced modeling vocabulary is also reflected in the basic application vocabulary introduced in Chapter 3.

Research Topic 4 is targeted by Chapter 5. The subject of this chapter is the design support for WIS requiring extensive user communication and collaboration. The design of such systems using only AM can appear to be difficult, because AM does not explicitly express the user tasks and their synchronization what makes the design less transparent and maintainable. We introduce a lightweight process model based on UML activity diagrams that explicitly defines the user tasks and their communication. An equally important contribution is the description of an automated procedure transforming this model to usable AM.

Research Topic 5 is discussed in Chapter 6. The technique of using template AM and their mapping to concrete domains aims at saving development efforts (and costs). The idea of this approach is based on classes of similar domains and it uses mapping from virtual template domain models to concrete ones. The eventual mapping issues in the process of deployment are discussed here.

Chapter 7 addresses Research Topic 6 by describing software prototypes developed for evaluating the application modeling techniques. The software includes a web server software that, based on a set of models, builds and populates web pages and processes user input at runtime. Next to it there are designer tools facilitating construction of domain and application model. Another tool helps to build mappings for application templates and their deployment.

The dissertation is concluded by Chapter 8. An important part of this conclusion is the question of possible directions of future research.
Chapter 2

Methods for Web Information System Design

In order to meet their business goals modern WIS must cope with severe requirements. First, there is typically a huge amount of dynamic data content that can be used by WIS that is possibly distributed and stored in different formats. Second, a WIS has typically a large and unknown number of users with different knowledge, interests and backgrounds. Third, modern WIS turn more and more from information providing systems to service providing systems. These requirements together with specifics of the Web systems boil down to the need for solid design methods and modeling techniques developed specifically for WIS design. This chapter summarizes existing approaches of WIS design and it focuses on the application logic specification. It addresses Research Topic 1 from Chapter 1. Selected WIS design methods RMM [Isakowitz et al., 1995], WebML [Ceri et al., 2003], OOHDM [Schwabe and Rossi, 1998], UWE [Koch et al., 2001], OO-H [Gomez and Cachero, 2003], and WSDM [De Troyer and Leune, 1998] are summarized and briefly evaluated. The conclusions drawn from the evaluation are used for rationalization of the extension of the Hera method described in this work. A summary of four methods can be found in [Barna et al., 2003].

Section 2.1 describes selected methods for WIS design. The Hera method is explained in Section 2.1.7 together with a summary of extensions that are discussed in detail in the rest of this dissertation. The chapter is concluded in Section 2.3.

2.1 Methods for WIS Design

2.1.1 Relationship Management Model (RMM) Method

Relationship Management Model [Isakowitz et al., 1995; Balasubramanian et al., 2001] is a one of the first modeling approaches developed for designing of navigation structure of data-intensive web applications. The RMM method distinguishes the domain design (originally
Methods for Web Information System Design

using Entity-Relationship model), the application (navigation) design user interface design, and implementation and testing phases. The domain design phase results in the domain model that is an E-R schema describing the structure of the domain data. The domain model is used in the application design phase, where an application model is built. The application model describes the navigation structure of the designed web application by rearranging concept relationships. In the user interface design phase the layout is specified and in the implementation phase an actual web application is generated and tested.

Specification of Application Logic

The navigation structure is defined by means of units representing groups of data attributes that are presented. Every unit (called slice) has a so called owner concept that is a domain concept (a concept defined in a domain model) and groups of selected data attributes of this concept and eventual data attributes of related domain concepts. Slices can be nested, where owner concepts of the parent and child slice should have defined relationship in the domain model. The extent of the child slice (set of instances) is based on the instance(s) of the owner concept of its parent slice and the domain relationship that connects both owner concepts. Links are represented by slice navigation relationships. Next to slices the RMM application model uses access structures that define the way how users can navigate to concrete slice instances. The two defined access structures are “guided tour” and “index”. RMM uses its own graphical notation.

RMM was one of the first methods that clearly separated navigation specification from the domain specification. The RMM application model defines only a navigation view on the domain data and it does not support any business logic specification. However, the navigation specification using the AM graphical notation is sufficient for actual (automated) generation of web pages (if we neglect a user interface specification). Due to its simplicity and manageability the RMM application model has motivated the navigation specification in Hera.

2.1.2 Web Modeling Language (WebML)

Method

The WebML method [Ceri et al., 2003] recognizes the data design phase, the hypertext design phase and the implementation phase. In the data design phase the domain model that defines the structure of WIS data is defined. WebML does not prescribe a particular data modeling technique and it is compatible with the E-R model and object-oriented models (for instance Unified Modeling Language, UML [Jacobson et al., 1999]). The output from the hypertext design phase is a hypertext model defining the navigation structure of WIS. The hypertext model distinguishes site views, pages, atomic units and their relationships. Pages are built from units and sites are built from pages. Data units display information about a single object (instance) and multidata units display information about multiple objects. Index units are the analogy to the index access structure, scroller units represent
another type of access structure for multiple objects (using scrolling mechanism). Data entry units facilitate gathering information from users. Content manipulation units are means of specification of system data state updates. They include units for adding, modification and removing of data objects and object relationships. Some units are aimed at invoking functionality of external systems by means of web service calls. The library of units is open and can be extended. Unit relationships (links) are contextual links that carry contextual parameters and also cause navigation from one unit to another. On the other hand navigation and transport links only transport parameters but do not cause navigation.

Specification of Application Logic

In WebML the presentation logic and business logic specification are encapsulated in the hypertext model. The business logic is represented by content manipulation units that are linked (by means of unit relationships) to units representing the navigation structure. Hence, the business logic and presentation logic specifications are integrated in a single model. However, for the design of WIS with complex processes (process-centric WIS in [Brambilla et al., 2006]) WebML has been extended with the process design and process distribution phases. The process model as the result of the process design phase expresses WIS functionality by means of hierarchically decomposed process(es). The process model is expressed using the BPMN [White, 2003] notation. In order to map this process model into a WebML hypertext model (and also for eventual implicit process modeling within hypertext model) few new types of units have been introduced, namely conditional navigation unit (for modeling of optional execution branches), activity start and end units, assignment units for work items, etc. In the process distribution phase designers assign sub-processes to different peers (servers) if the designed WIS relies on a distributed environment. An explicit process model can automatically be mapped to a skeleton of the hypertext model.

2.1.3 Object-Oriented Hypermedia Design Model (OOHDM)

Method

Object-Oriented Hypermedia Design Model [Schwabe et al., 1996; Schwabe and Rossi, 1998] is a WIS design method based on the object-oriented design paradigm. The method distinguishes between the following design phases: requirement specification, conceptual design, navigation design, abstract interface design and implementation. Gathering and formulating requirements is a common activity that takes place in a design process using any method. The requirements describe desired (functional and non-functional) features of the designed system. These requirements are reflected in the conceptual design (here it covers the domain and the application design) where the structure of the domain is defined using UML diagrams. The navigation structure is defined by the navigation class schema and the navigation context schema. The abstract interface design extends the
notion of navigation structure with abstract data views (ADV) that define the appearance of navigation classes (attributes) and add interface objects such as buttons, selection lists, or menus to the navigation models. ADV charts are specializations of UML state charts and they model dynamic properties of the interface (input, output and perceived events triggered from the interface elements).

Specification of Application Logic

The application logic in OOHDM is defined in the conceptual model, navigation model and abstract interface model. The navigation model consists of (a set of) navigation class schemata specifying navigation classes, access structures defining the way how navigation classes can be accessed and navigation relationships representing links. The navigation context schema determines the grouping of the navigation space by specifying how navigation classes, access structures and links are instantiated within different navigation contexts.

The navigation structure is determined by the navigation model, where its dynamic properties (instantiation decided at runtime) are defined in the navigation context schema. ADV charts specify responses to events invoked by user actions. The business logic is encapsulated in methods of conceptual and navigation classes that are called on interface events.

2.1.4 UML-Based Web Engineering Method (UWE)

Method

UML-Based Web Engineering Method UWE [Koch et al., 2001] is an object-oriented WIS design approach. This method is based on the unified software process (USP) and its models are extensions of standard UML models. All extensions are expressed using generic meta-classes. The design phase includes the requirement analysis and gathering, conceptual design, navigation design, presentation design and testing and implementation. Requirements are collected and stored in the form of UML use case diagrams. The application domain data structure is specified in a conceptual model that is a standard UML class diagram. The navigation design phase delivers navigation models, concretely navigation space model and navigation structure model. The former one defines objects that can be visited by users in the course of navigation (what objects can be browsed), whereas the later one defines the structure of navigation itself (how the objects are browsed). The presentation model built in the presentation design phase is also a class diagram, where compositions (aggregations) of classes are denoted as visual compositions (for instance attributes inside a navigation class). Moreover these sketches (description of the page layout) determine the relative position (layout) of visual elements. A skeleton of the application can be generated from the model that is finalized and made executable in the implementation phase.
2.1 Methods for WIS Design

Specification of Application Logic

The user interface (navigation structure) is described in a navigation space model and a navigation structure model. The dynamic aspects of the application are specified using UML statechart diagrams, UML sequence diagrams, UML collaboration diagrams and UML activity diagrams. UML statechart diagrams describe presentation state transitions where the states are associated with presentation classes and state transitions with triggering events restricted by guarding conditions. UML sequence diagrams describe scenarios of particular use cases in temporal order (invoking of class methods). UML collaboration diagrams can be used in a similar way as statecharts, but they also distinguish between different (user) roles and their relationships. UML activity diagrams model tasks that are performed in the system and that correspond to previously defined use cases. The tasks are modeled as sequences of activity states, synchronization primitives and transitions between them.

Modeling business process using UWE (and OO-H) is described in [Koch et al., 2004]. Process classes defined in the process structure model are linked to navigation classes in navigation structure models. The process structure model is derived from the conceptual model. The process workflow is specified in a process flow model and it is expressed using (set of) UML activity diagrams showing the sequencing of activity states and their relationships to process classes. The process models can be used for automatic generation of the application skeleton. In UWE the application model integrates the navigation structure specification with the process specification.

Besides the general-purpose process specification there is a personalization framework dedicated to the UWE method described in [Koch and Wirsing, 2002]. This framework adds a notion of a user model together with user model meta-classes and meta-classes for defining personalization rules. These personalization rules are based on the event-condition-action paradigm and they can be triggered by events from the web interface. The rules can update the navigation structure and data values in the user model data space.

2.1.5 Object-Oriented Hypermedia Model (OO-H)

Method

Object-Oriented Hypermedia Model [Gomez and Cachero, 2003] is another representative of the object-oriented design approaches. It recognizes similar design phases as the UWE method. In the requirement specification phase use cases are described using UML use case diagrams. The output of the conceptual design phase is a set of UML class diagrams constituting the conceptual (domain) model. Thought based on UML the OO-H navigation model built within the application design phase differs from the one used in UWE. The navigation model is defined in the form of a navigation access diagram (NAD) specifying the structure and dynamic properties of the application web interface.
Specification of Application Logic

The presentation logic is expressed in NAD that contains navigation classes derived from conceptual classes, navigation links and complex navigation targets representing complete subsystems. Navigation links are important for the specification of dynamic properties of the application web interface. They can have multiple characteristics attached to them as are for instance the link type (requirement links for initiating navigation, traversal links for regular navigation and service links for invoking some business or presentation logic), activating agents (links can be associated with particular users or systems available only to them), navigation effects (the link presents only an aggregation of classes or real navigation), or filters defined using OCL.

The OO-H support of process-driven design is discussed in [Koch et al., 2004]. In OO-H for process modeling UML activity diagrams are used. Activity states can have stereotypes <<Transactional>> and <<non Transactional>> where the former one denotes activities that change the system data state (manipulate system data) and the later one denotes activities that do not change the system data state. An OO-H business process model can be used for automatic generation of the navigation model (NAD).

Similarly to how it is in UWE, the OO-H method has its own personalization framework that consists of the user and the personalization meta-model and appropriate software tools. The user model (specialization of the user meta-model) describes the structure of information about users and the personalization model (specialization of the personalization meta-model) represents the personalization effects - (dynamic) adaptation of navigation structure and updates of user related information.

2.1.6 Web Site Design Method (WSDM)

Method

Web Site Design Method [De Troyer and Leune, 1998] is an audience-driven approach for modeling and design of WIS. The key and initial point of the design is the mission statement. The mission statement characterizes the purpose, the subject and the audience of the designed WIS. The audience is partitioned into user classes. The same audience classes share the same functionality of the same access to information. In the conceptual design phase the tasks and navigation tracks for user classes are modeled, unlike in some other methods where the domain structure is defined first. Tasks are hierarchically organized using concurrent task trees and object chunks (specification of portions of information used in tasks). The navigation model is a graph in which the edges can be links or aggregation relationships. The whole navigation model consists of navigation tracks for every user class. The implementation phase distinguishes page design (mapping of navigation nodes to pages, adding of adaptation), presentation design (specification of presentation layout and style), and data design (specification of overall data schema based on object chunks).
Specification of Application Logic

The specification of presentation logic is expressed in the navigation model referring to domain data by means of object chunks. The business logic of the system is expressed in a task model consisting of concurrent task trees (CTT). A CTT is a task model where the task hierarchy expresses the structural decomposition of tasks. Tasks have different types (user, application, interaction, and abstract) and there are temporal relationships between them allowing to define task synchronization. Object chunks associated with all defined tasks characterize information that is used or presented within a particular task. WSDM keeps the business logic specification well separated from the specification of presentation logic, where the former one is captured in the task model and the latter in the navigation model. Modeling complex business processes using WSDM is discussed in [De Troyer and Casteleyn, 2003].

2.1.7 The Hera Method

Method

The Hera method is one of the very first methods that used a Semantic Web language for specifying WIS. It is a model-driven method, which means that developed WIS are (fully) specified by a set of models. The structure of a data domain is defined in the domain (or conceptual) model. The navigation structure is expressed by an application model built on top of the domain model. The application model proposal was motivated by the RMM method described before. The presentation model extends the application model with the specification of presentation pages layout information. All models are expressed in RDFS [Brickley and Guha, 2004] and intermediate data (instantiations of a model) are in RDF [Lassila and Swick, 1999] format. The design process is similarly divided into domain, application and presentation design. The Hera hosting platform is divided accordingly into three layers that provide runtime or non-runtime presentation instantiation: the semantic layer that provides data retrieval, the application layer that populates the application model (its parts specifying single presentation pages) and the presentation layer that produces pages in the end-format (e.g. HTML).

Specification of Application Logic

The application model (AM) in the early Hera method specifies the structure of a hypermedia presentation for semantically annotated domain data. The navigation nodes are abstracted by the so called slice concept. A slice as the basic building block of AM represents a chunk of information related to a domain concept (and its related concepts) and presented in the resulting hypermedia presentation. It also defines how the concrete instances of the concept(s) are retrieved. Slices can be composed in a modular way and slices that are not part of other slices (top-level slices) represent whole pages. Although the original AM described only navigation logic over a data domain, its clear and modular structure offered an excellent foundation for its further extensions.
2.2 Hera Extension

This section summarizes the extension of the Hera method discussed in detail in the further chapters of this dissertation. The original proposal (Frasincar et al., 2001, 2002) summarized in Section 2.1.7 underwent an evolution process that made the method a full-fledged WIS design approach (a good overview of the Hera evolution, its architecture, and implementation issues can be found in Frasincar et al., 2006; Rutten et al., 2004). The extension of the method discussed here is summarized by five steps in Chapter 1, Section 1.2. The next paragraphs refer to these steps while explaining the extension that was designed for and applied to the Hera models and Hera architecture.

2.2.1 Extended Hera Architecture

The Hera runtime environment is organized into the three layers mentioned previously (the semantic layer, the application layer, and the presentation layer). The architecture of a Hera WIS using the models for its operation is shown in Figure 2.1.

The advanced AM (defined within Step 2) allows WIS to perform data content updates triggered by user actions. The updated data content is used for populating subsequently visited pages. For this reason it is necessary that data content manipulations are executed as soon as possible (as soon as they are triggered) and that visited pages are populated with data as late as possible (when they are requested). This approach ensuring fresh data presentation has consequences on the Hera system architecture. The three layer architecture processes the client requests at runtime, performs eventual updates and retrieves data needed for the requested page. It is impossible to use the original approach, where the whole hypermedia presentation was built at once prior to its deployment.

A typical scenario of the user interaction begins with a (http) request coming from an end-user’s Web client. This request is processed in the application layer and depending on the AM specification it is transformed into a retrieval request that is then passed to the semantic layer and possibly also into a data transaction request performed within the application layer (prior to or after the retrieval). Data retrieval requests processed by the semantic layer deliver data needed for the construction of a requested page. The semantic layer can be seen as a database wrapper that provides the application layer with a uniform query interface to data sources. The retrieved data is used by the application layer for building the instance of the AM element (unit) representing the requested page that is further transformed into the end format (e.g. HTML) in the presentation layer.

2.2.2 Extended Hera Models

As we mentioned in the previous text the design of advanced WIS requires not only the specification of a static navigation structure but also of its dynamic updates as well as the specification of data content updates. In addition, the data content manipulation must be synchronized with the web interface in both directions: the presented data must be
up-to-date and the user feedback should be monitored and eventually reflected in the data content.

The following specification facilities have been added to the application model:

- **Data retrieval facilities** allowing a more flexible specification of presented content (covering Step 1 in Chapter 1, Section 1.2) discussed in Chapter 3; compared to the original Hera AM an arbitrary query can be associated to links and navigation units.

- **Data update operations** allowing data manipulations. The data update operations are expressed by means of event-triggered (Event-Condition-Action) rules that constitute a content management sub-model of the application model (addressing Step 2 in Chapter 1, Section 1.2). These operations are discussed in Chapter 4.

- **Entry forms** allowing gathering of information from users (addressing Step 2 in Chapter 1, Section 1.2) are discussed in Chapter 4.

- **Inclusion conditions** and **optional navigation paths** providing additional means needed for dynamic updates of the navigation structure (addressing Step 2 in Chapter 1, Section 1.2) are discussed in Chapter 4.

Supportive models for the design of advanced WIS requiring explicit specification of user collaboration and communication are:

- A lightweight **business process model** and **task model**. The first model specifies the life cycle of business items (e.g. of conference submissions exchanged in the review process between authors, committee members and reviewers) partitioned into tasks of different types of users. The second model maps the user types to the tasks. Both models are introduced within Step 3 in Chapter 1, Section 1.2

- **AM templates** that are in fact regular AM (or their parts) defined on template (virtual) domain models. By proper mapping of virtual domain concepts to real ones
one can deploy a template AM to a concrete domain. This proposal is part of Step 4 in Chapter 1, Section 1.2.

Step 5 in Chapter 1, Section 1.2 covers development of software prototypes that help to validate the proposed model extension. The whole software development framework for Hera is shown in Figure 2.2. The original set of tools for basic modeling includes DM Builder, AM Builder and PM Builder and has been extended with tools facilitating the design of AM. The Mapping Builder helps with the specification of mappings needed for deployment of application templates, the WM Builder is a tool supporting the development of workflow models in a visual way. The AT-AM Transformer (application template to application model) allows deployment of application templates to concrete domains and the WM-AM Transformer (workflow model to application model) generates skeletons of AM from workflow models.

2.3 Conclusion

This chapter addresses Research Topic 1 through a brief discussion of selected methods for WIS design while focusing on the application logic specification. The particular aspects of the Hera extension are summarized and they are shown in the context of Hera framework. The extended Hera method combines the simplicity of application modeling inherited from RMM with powerful specification means and the ability to cope with the Semantic Web environment.
Chapter 3

Domain and Basic Application Modeling

This chapter introduces basic modeling terms used in the domain and application modeling using Hera. The basic application model is addressed in Research Topic 2 from Chapter 1 and is a subset of the advanced (full) application model. It describes only the navigation structure of WIS and data retrieval facilities associated with the navigation structure. It does not specify the data transactions and other advanced facilities discussed in Chapter 4. The domain model reflects the structure of data content used in web applications and the application model determines the navigation view on the data content.

All Hera models are implemented as RDFS [Brickley and Guha, 2004] vocabularies. The choice of RDFS allows a seamless design of SW-ready applications and makes all existing applications open for future extensions. All modeling primitives are however defined in a way not restricting their possible implementation using a different modeling paradigm.

Section 3.1 introduces basic terminology used for the specification of the domain data content. Formal definitions are used where this contributes to clarity and accuracy of the text. A number of small examples based on a sample domain described in Section 3.1 support the formal or textual explanations. Section 3.2 defines a basic application model as a (static) navigation view on the domain data.

3.1 Domain Modeling

Structuring data content used in web applications is one of the key steps in WIS design. All Hera models use this domain specification and build on it. The basic blocks of the domain structure specification are intentionally defined in a way that imposes as few restrictions on the target specification technique as possible. However, all Hera models are implemented as RDFS vocabularies that are more prescriptive than restrictive.

Concrete domain data consist of resources, resource relationships and data values. The structure of a domain is described by means of resource types (concepts), resource relationship types (properties) and data value types (attributes). These terms are selected
such that they have counterparts in different modeling approaches. The term “resource” corresponds to “entity” in entity-relationship modeling (ER) [Thalheim, 2000; Abiteboul et al., 1995], to “object” in an object-oriented modeling method (consider for example UML [Jacobson et al., 1999]) and to “resource” in RDF [Lassila and Swick, 1999]. The term “concept” can be compared to “entity set” in ER, “class” in UML, or “RDFS concept”. An important concept property improving the expressiveness of models is the specialization property (sub-concept and sub-property) allowing to define specialized concepts (adapted for more special use) based on a parent concept(s). The specialization is a transitive property and it is used for the hierarchical structuring of models and for reuse. It is included in all the mentioned modeling approaches.

The sample domain vocabulary shown in Figure 3.1 represents a model of a conference submission system. The concepts representing involved human actors as Author, Reviewer and PCMember are derived from (are sub-concepts of) the general Person concept. Concepts like Paper, Review, Evaluation, Conference and Track have self-explanatory names. A number of properties express semantic relationships between the concepts that will be used further. This domain is used in the whole dissertation as a running example for demonstrating different modeling techniques.

Let us now define and explain elementary terms we use in domain modeling. The most important terms are domain, vocabulary and domain model. Later we introduce special properties used in domain modeling.
3.1 Domain Modeling

Definition 3.1 (Domain) A domain $\text{Dom}$ is a tuple $< R, \text{Rel}, D, L >$, where:

- $R$ is a finite set of data resources in $\text{Dom}$,
- $\text{Rel} = \{ p | p \subseteq R \times R \}$ is a finite set of binary relations, where every member relation represents a resource relationship in $\text{Dom}$,
- $D$ is a finite set of all data values in $\text{Dom}$ and
- $L = \{ l | l \subseteq R \times D \}$ is a finite set of binary relations, where every member relation represents a resource-value mapping in $\text{Dom}$.

Example 3.1 (Domain) An excerpt of the domain containing the data about papers and authors is graphically presented in Figure 3.2. Ovals represent resources, arrows represent resource relationships and literals, and rectangles represent data values. The example domain has the following values:

- $R = \{ \text{Paper}_1, \text{Author}_1, \text{Author}_2 \}$
- $\text{Rel} = \{ \text{paper2author} \}$
- $\text{paper2author} = \{ (\text{Paper}_1, \text{Author}_1), (\text{Paper}_1, \text{Author}_2) \}$
- $D = \{ "WIS Modeling", "John Smith", "Carla Haul" \}$
- $L = \{ \text{concept2text} \}$
- $\text{concept2text} = \{ (\text{Paper}_1, "WIS Modeling"), (\text{Author}_1, "John Smith"), (\text{Author}_2, "Carla Haul") \}$

A domain is characterized by its data content. It has a simple structure distinguishing resources, their relationships and data values. Note that the notion of resources, relationships and values does not imply any higher degree of data organization and thus it is difficult to specify any structural data retrieval method (potentially all resources can have different semantics and information structure). The notion of resource and relationship types that would group them, based on structural and semantical similarities, is introduced with the definition of a domain vocabulary.
Definition 3.2 (Vocabulary) A Vocabulary \( \text{Voc} \) is the tuple \(< C, P, T, A >\) where:

- \( C \) is a finite set of resource types (concepts) in \( \text{Voc} \), where a concept \( c \in C \) is a set of resources (of the type \( c \))
- \( P = \{ p | p \subseteq C \times C \} \) is a finite set of binary relations defining concept properties in \( \text{Voc} \)
- \( T \) is a finite set of data types in \( \text{Voc} \)
- \( A = \{ a | a \subseteq C \times T \} \) is a finite set of binary relations defining concept attributes in \( \text{Voc} \)

Example 3.2 (Vocabulary) Figure 3.3 shows a graph representing a vocabulary with concepts Paper and Author, concept property written_by and literal properties title and name. Ovals denote concepts, rectangles denote data types, and arrows denote concept properties and literals. The tuple representation of this vocabulary is:

\[
\begin{align*}
C &= \{ \text{Paper, Author} \} \\
P &= \{ \text{written_by} \} \\
\text{written_by} &= \{ (\text{Paper, Author}) \} \\
A &= \{ \text{title, name} \} \\
\text{title} &= \{ (\text{Paper, String}) \} \\
\text{name} &= \{ (\text{Author, String}) \} \\
T &= \{ \text{String} \}
\end{align*}
\]

Figure 3.3: An example of a vocabulary

The notion of vocabulary allows abstraction from the level of concrete data resources (relationships and values) and allows to define data access to an arbitrary domain that can be described by a vocabulary. Note that because the set of properties \( P \) in Definition 3.2 is a set of relations a single property \( p \in P \) can possibly define relationships between multiple concepts (as it is for instance in RDFS). Similarly, a single attribute \( a \in A \) can be an attribute of multiple concepts. It is up to domain designers to explore the benefits of this relaxed approach and to avoid unwanted overlapping by using proper names.

Vocabularies group resources and their relationships into concepts (resource types) and properties (resource relationship types). A set of related concepts and properties describes
the structure of a domain and therefore it can be seen as a vocabulary of domain terms. Concrete domain data represented by resources and their relationships are then usually partitioned by the defined domain vocabulary terms. Definition 3.3 introduces the type concept property needed for formal specification of relationships between domains and vocabularies given in Definition 3.5.

**Definition 3.3 (type property)** Consider a vocabulary \( \text{Voc} = \langle C, P, T, A \rangle \) and a domain \( \text{Dom} = \langle R, \text{Rel}, D, L \rangle \). The relation \( \text{type} \subseteq (R \times C) \cup (\text{Rel} \times P) \) determines the relationship between resources (relationships) in a domain \( \text{Dom} \) and concepts (properties) in a vocabulary \( \text{Voc} \).

The specification of the type property depends on the domain design and it allows grouping of domain resources and relationships into concepts and properties based on structural and semantical similarity. Another special property \( \text{datatype} \) determines associate data values from domains with data types in vocabularies.

**Definition 3.4 (datatype property)** Consider a vocabulary \( \text{Voc} = \langle C, P, T, A \rangle \) and a domain \( \text{Dom} = \langle R, \text{Rel}, D, L \rangle \). The relation \( \text{datatype} \subseteq (D \times T) \) determines data types defined in \( \text{Voc} \) of data values from \( \text{Dom} \).

For domain modeling it is important to have some criteria deciding whether a vocabulary really specifies a concrete domain and vice-versa. Different modeling methods have different views on how rigorous these criteria are. Prescriptive models, as it is with for instance the ER model, do not allow any structural deviations of domain data from their vocabulary (schema), whereas more descriptive models like RDFS allow them. We consider the criteria in Definition 3.5 that rely on the type and datatype relations.

**Definition 3.5 (Domain Model)** A vocabulary \( \text{Voc} = \langle C, P, T, A \rangle \) is a domain model of a domain \( \text{Dom} = \langle R, \text{Rel}, D, L \rangle \) if the following properties hold:

- \( \forall r \in R : \text{type}(r, c) \land c \in C \); all resources in the domain \( \text{Dom} \) belong to concepts that exist in the vocabulary \( \text{Voc} \),

- \( \forall \text{rel} \in \text{Rel} : \text{type}(\text{rel}, p) \land p \in P \); all resource relationships in the domain \( \text{Dom} \) are present in the vocabulary \( \text{Voc} \).

This fact can be written compactly as \( \text{Voc} = D_M(\text{Dom}) \) or \( \text{Dom} = I(\text{Voc}) \) (\( \text{Dom} \) is an interpretation of \( \text{Voc} \)).

The relationship between domain models and domains as defined here is rather lightweight. The domain model does not impose strict structural restrictions to its domain. This relaxed approach similar to the RDFS model makes use of a prescriptive description rather than a restrictive one like for instance in the ER model. It shifts more responsibility to domain (and application) designers, but it makes data retrieval facilities simpler (no need for strict structural checks). However, it is possible to make the model more rigid by defining additional restrictions if needed.
In order to simplify our notation the following operators, that represent often used features of concepts and properties, are defined:

- $\text{Dom}(p) = \{c| (c, x) \in p, c \in C\}$ gives a set of all domain concepts of a property $p \in P$,
- $\text{Ran}(p) = \{c| (x, c) \in p, c \in C\}$ gives a set of all range concepts of a property $p \in P$,
- $\text{Prop}(c) = \{(c, x)| (c, x) \in p\}$ gives a set of all properties of a concept $c \in C$,
- $\text{Attr}(c) = \{(c, x)| (c, x) \in A\}$ gives a set of all attributes of a concept $c \in C$.

One of the most used reuse mechanisms in conceptual modeling is specialization (or its inverse approach: generalization) of concepts and properties. The specialization is a transitive binary relation between concepts (properties) defined by means of sub-concept relation.

**Definition 3.6 (sub-concept)** Let $\mathbf{Voc}$ be a vocabulary $< C, P, T, A >$. Let $\bigcup I(\mathbf{Voc})$ be a set of all interpretations of the vocabulary $\mathbf{Voc}$, where all possible interpretations (domains) $I(\mathbf{Voc})$ are in the form $< R, \text{Rel}, D, L >$. The sub-concept relation $c_1 \preceq c_2$ is defined for two concepts $c_1, c_2 \in C$ if $\forall r \in R, \text{type}(r, c_1) \Rightarrow \text{type}(r, c_2)$.

In words it can be said that a concept $c_1$ is a sub-concept of a concept $c_2$ if resources — in all possible domains corresponding to the vocabulary the concepts ($c_1$ and $c_2$) belong to — that are of the type $c_1$ are also of the type $c_2$. It is easy to show that the sub-concept relation is reflexive ($c \preceq c$), it is also antisymmetric ($c_1 \preceq c_2$ does not imply $c_2 \preceq c_1$) and it is also transitive (if $c_1 \preceq c_2$ and $c_2 \preceq c_3$ then also $c_1 \preceq c_3$). A similar relation can be defined for properties.

**Definition 3.7 (sub-property)** Let $\mathbf{Voc}$ be a vocabulary $< C, P, T, A >$. Let $\bigcup I(\mathbf{Voc})$ be a set of all interpretations of the vocabulary $\mathbf{Voc}$ where all possible interpretations (domains) $I(\mathbf{Voc})$ are in the form $< R, \text{Rel}, D, L >$. The sub-property relation $p_1 \preceq_P p_2$ is defined for two properties $p_1, p_2 \in P$ if $\forall r \in \text{Rel}, \text{type}(r, p_1) \Rightarrow \text{type}(r, p_2)$.

Definitions 3.6 and 3.7 impose the necessary condition for existence of the sub-concept and sub-property relations. For modeling it is practical (and common) to include the following structural restrictions:

- $c_1 \preceq c_2 \Rightarrow \text{Prop}(c_1) \subseteq \text{Prop}(c_2) \land \text{Attr}(c_1) \subseteq \text{Attr}(c_2)$
- $p_1 \preceq_P p_2 \Rightarrow \text{Ran}(p_1) \subseteq \text{Ran}(p_2) \land \text{Dom}(p_1) \subseteq \text{Dom}(p_2)$

These simple expressions describe the fact that concepts inherit attributes and properties from their super-concepts and properties inherit domains and ranges from their super-properties.
3.2 Basic Application Modeling

The application modeling can be seen as a special case of the domain modeling. Application models target the domain of software (Web) designs. Every design method can (and typically does) have its own vocabulary of modeling terms. In this section the basic building blocks of Hera application models are defined. These building blocks are called “basic” because they enable us to build models that define navigation views on the domain data but do not allow the specification of a more complex application behavior (e.g. the specification of effects of user actions on the domain data).

The approach presented in this section extends the first Hera AM presented in [Frasincar et al., 2002]. In summary, application models (AM) define the business logic and partly the presentation logic of Web applications. They are specified using application model terms based on domain concepts and properties. Basic types of AM terms are:

- **Data accessors** (queries) specifying how the presented data are retrieved from the domain,

- **Navigation units** (NU) based on data accessors grouping presented information with navigation facilities and

- **Links** specifying navigation facilities by means of hyperlinking of NU.

Conceptually, a navigation view represented by AM is a restructured domain model with navigation facilities (links). New navigation concepts (navigation units) and navigation properties are derived by means of specifying data access to domain databases and by means of adding navigation facilities form a basic AM.

There are multiple ways of how to express NU and their relationships. The approach discussed in this chapter extends the technique explained in the previous chapter with properties defining the composition of NU and the addition of NU attributes. NU can be composed from simpler NU, links, and attributes by means of composition properties explained in further text. All building elements of AM except simple concept attributes are built in a modular way allowing their reuse within different NU.

As mentioned previously, all models in Hera discussed in this dissertation are represented through RDFS vocabularies. For clarity reasons some examples of models are expressed using an abstract syntax briefly discussed in the following paragraph. Building blocks of models are formally defined, but their RDFS representation is the subject of Chapter 4.

All AM are expressed as RDFS vocabularies. Although using RDFS brings numerous advantages (inter-operability, SW readiness, possibility to verify model consistency and correctness etc.), an RDFS representation of a concrete AM may not be clear and readable for a human audience. Wherever the graphical notation does not suffice to cover all needed details we use an abstract syntax (AS) notation.
3.2.1 Data Accessors

Data accessors are data retrieval queries used in AM. They determine the data content of web applications. The retrieval queries are based on the following operations:

- Data projection that specifies a subset of (retrieved) domain data based on naming a set of required vocabulary terms (concepts and properties). In other words, projection specifies the retrieved data based on choosing concepts.

- Data selection that specifies a subset of (retrieved) domain data based on constraining the set of instances of domain vocabulary terms. In other words, selection specifies the retrieved data by conditional expressions containing references to the data content.

- Set operators (optionally) union, difference, and intersections that can be applied on queries (based on projection and selection).

Graph patterns are basic building blocks of the query language applied to the graph data model used in Hera. Graph patterns are expressions that can be evaluated using concrete (domain) data as to be true or false. In this manner a set of matching sub-graphs can be selected. Graph patterns define both data projection and selection operations.

Graph patterns that are basic building blocks of queries are built from path expressions that are elementary graph patterns. The syntax of path expressions is inspired by the SeRQL RDFS query language.

**Definition 3.8 (Data Path)** Consider a domain vocabulary $\text{Voc}=< C, P, T, A >$. A data path is an expression in one of the forms:

\[
\{c_1, B(c_1)\}p_1, B(p_1)\{c_2, B(c_2)\}p_2, B(p_2)...\{c_n, B(c_n)\} \quad (3.2.1)
\]

\[
\{c_1, B(c_1)\}p_1, B(p_1)\{c_2, B(c_2)\}p_2, B(p_2)...\{c_n, B(c_n)\}l_n, B(l_n) \quad (3.2.2)
\]

where $c_1, ...c_n \in C$ and $p_1, ...p_n \in P$. For all concepts and properties in $\pi$ holds: $\forall c_i, p_i, c_{i+1} : c_i \in \text{Dom}(p_i) \land c_{i+1} \in \text{Ran}(p_i)$ and $l_n \in \text{Attr}(c_n)$. The optional $B()$ expressions are binding expressions for concepts/properties containing variables bound to a concrete concept or a property.

Binding variables can be in a data path context assigned to concept or attribute values. The binding variables are also used for the construction of graph patterns. A data path itself is an elementary graph pattern and it can be matched with data graphs.

**Example 3.3 (Data Path)** Consider the sample domain introduced previously. The following data path associates reviews with reviewers.

\[
\{\text{cm:Review AS E}\}<\text{cm:madeBy}{\{\text{cm:Reviewer AS R}\}}
\]
Definition 3.9 (Graph Pattern) A graph pattern $\Pi$ is a non-empty set of data paths 
$\{\pi_i | i = 1...n\}$ that are in the form described in Definition 3.8.

Example 3.4 (Graph Pattern) Consider the sample domain introduced in the previous chapter. The following GP retrieves PC members that have assigned themselves as reviewers.

$$\{cm:Review AS E}\langle cm:madeBy\rangle\{cm:Reviewer AS R\},$$

$$\{E\}\langle assignedBy\rangle\{cm:PCMember AS P\}$$

Note the binding expressions determined by the AS keyword. The binding variable E shared by both data paths in the GP indicates that the matching data graph for both paths will contain the same shared set of cm:Reviewer instances.

Note that a single concept name is also a data path (with length 1) and thus also a graph pattern. Graph patterns are important for building data queries. A graph pattern is composed of a set of paths that can be independent or dependent (by using shared concepts). Queries contain a projection and selection part. Queries can be also composed from other queries by applying set operators to result sets of the partial queries.

Definition 3.10 (Retrieval Query) A retrieval query is a tuple $q = \langle \pi, \sigma, \omega \rangle$ or a set expression $S$ containing selection queries of type $q$ and set operators where:

- $\pi$ is data projection represented by a graph pattern (graph query) or a list of expressions containing binding variables from $\sigma$ (table query),
- $\sigma$ is data selection represented by a graph pattern,
- $\omega$ is an optional ordering expression.

A retrieval query is also an expression on queries of the type $Q$ containing binary operators “$\cup$” (union), “$\cap$” (intersection), and “$-$” (difference).

Example 3.5 (Retrieval Query) Consider the sample domain described in Section 3.1. The query QueryUnion retrieves all authors that have papers accepted together with reviewers that have reviewed accepted papers ordered by surnames.

QUERY QueryAuthors=

```
SELECT \{ A \} FROM
\{ {dm:Author AS A}\langle dm:wrote\rangle\{dm:Paper AS P\}; 
{dm:Track}\langle dm:accepted\rangle\{P\} 
\}
WHERE \{ A.cid = P.cid \}
```

QUERY QueryReviewers =
{
    SELECT { R }
    FROM
    {
        {dm:Reviewer AS R}<dm:makes>{dm:Review}<dm:inEval>
        {dm:Evaluation}<dm:ofPaper>{dm:Paper AS P},
        {dm:Track}<dm:accepted>{P}
    }
    WHERE { R.cid = P.cid }
}
QUERY QueryUnion = {QueryAuthors UNION QueryReviewers} }

Note that the structure of queries and graph patterns is similar to SeRQL, an RDFS query language. In fact, they represent an extension of these query languages. The existing software implementation of the Hera system described later in this dissertation uses the extended SeRQL interface for data content retrieval and data content manipulation.

3.2.2 Navigation Units

AM consists of NU and their relationships. NU are containers of attributes, links and child units. NU are either context-free — meaning that they are not associated with any particular data concepts — or they are based on a root concept. The selection of NU instances is specified by the root query and is often parameterized by the NU context (for instance by the content of an incoming link). The AM is defined as follows:

- **NU** represent navigation nodes and determine their content,
- **NU relationships** can be of the types:
  - Compositions define the content of NU by means of their composition and data restrictions for the child NU,
  - Links define the navigational relationships between NU and data restrictions for the destination NU,
- **Attributes** are atomic NU (they do not have any relationships) that define the data content presented in their parent NU.

For the specification of the AM composition and its effects on the retrieved data content it is useful to introduce a parameterized query. A parameterized query is a query containing variables that can be bound outside its scope. Such a query behaves as a function that gives results depending not only on the state of database but also on assigned values of its parameters.
Definition 3.11 (Parameterized Query) Parameterized query is defined as a tuple $Q(P) = \langle q, P \rangle$ where $q$ is a query and $P$ is a set of selected variables appearing in the projections and/or selections of $Q$ (set of parameters). The set $P$ is called the parameter set and its members can be bound outside the query $q$.

Example 3.6 (Parameterized Query) Consider the sample domain described previously. The query QueryAuthors retrieves authors with a surname given by the string parameter $S$ that have their papers accepted.

QUERY QueryAuthors(xsd:String S)=
  {
    SELECT { A }
    FROM
    {
      {dm:Author AS A}<dm:wrote>{dm:Paper AS P},
      <dm:sname>{N};
      {dm:Track}<dm:accepted>{P}
    }
    WHERE { N LIKE S }
  }

Consider now NU as essential building blocks of the AM. A NU is a container of other NU and atomic values that specify how the content of navigation nodes is created (retrieved or computed). They must allow modular composition supporting the retrieval semantics that determines how the content of composed NU is computed. In order to facilitate such composition NU have a uniform interface that is based on a single domain concept (root concept) and an expression specifying a set of input instances for a concrete NU in a concrete context.

Definition 3.12 (NU) A navigation unit is a tuple $NU = \langle r, q, C, L, A \rangle$ where:

- $r$ is the root concept of the domain model,
- $q$ is a (optionally parameterized) retrieval query returning a set of instances of the root concept,
- $C$ is a set of compositions used in $NU$,
- $L$ is a set of links used in $NU$ and
- $A$ is the set of attributes in $NU$.

A NU specifies the selection of NU instances of its root concept. Attributes define what information is presented. The content of a NU is specified further using compositions and links. Attributes, links and compositions are explained later in this chapter. The root query of a NU is denoted as $root(NU)$. 
Example 3.7 (NU) Consider a NU that presents information about authors of the selected papers, then it will be:

\[
\text{NU Author.Accepted} = \\
\{ \\
\text{ROOT}(\text{dm:Paper P}) = \\
\{ \\
\text{SELECT \{ A \} FROM} \\
\{ \\
\text{\{dm:Author AS A\}<dm:wrote>{dm:Paper AS P}}, \\
\text{\{dm:Track}\<dm:accepted>{P}} \\
\} \\
\} \\
\ldots \\
\}
\]

The \text{ROOT} keyword defines the root query retrieving authors of paper(s) given by the parameter \text{P}. The root concept of the NU (\text{r} in Definition 3.12) is \text{dm:Paper}. The set of authors \text{A} can be further used in possible nested NU.

A. Compositions

The NU composition relationship is a powerful means of constructing complex NU. It specifies the content structure of NU by means of specifying aggregation relation between units. In addition, the aggregation is enriched with data retrieval semantics in order to specify how the data retrieval restrictions are propagated from parent to child units.

\textbf{Definition 3.13 (Composition)} A composition \text{comp} \subseteq NU \times NU is a NU relationship that specifies modular construction of NU.

Note that the basic composition is defined as a relationship without any restriction. It is typically defined by the application designer in order to specify the modular construction of navigation units. However, the composition relationship does not define any data restriction applied to the parent unit and child unit extents. Data compositions add the ability to define these restrictions and thus facilitate the modular application model construction.

\textbf{Definition 3.14 (Data Composition)} A data composition is a tuple \text{dcomp} = < \text{C}, \text{Q} >, where \text{C} \subseteq NU \times NU is a composition relationship and \text{Q} is a parameterized data retrieval query. If \text{C}(\text{nu}_P, \text{nu}_C) then \text{Q} defines the extent (data instances) of \text{root(\text{nu}_C)} and at the same time the input data set of \text{Q} is the extent of \text{root(\text{nu}_P)}.

This mechanism allows the propagation of data restrictions from parent to child units. In this way complex units can be built from simpler ones. Moreover, units can be reused within different contexts.
Example 3.8 (Data Composition) The following composition track2author defines a relationship between the Track.Details NU presenting a track's information and the Author.Name NU presenting an information about authors that published their papers in the presented track. The query QueryTrack2Authors takes the instance(s) of the dm:Track root concept as a parameter. The projection of this query is the dm:Author concept and it retrieves a set of authors that published in the input track(s) used in the composition as the input set for the Authors.Name NU instantiation.

```
NU Track.Details( dm:Track AS T ) =
{
    ROOT { T }
    ...
    DCOMP track2author =
    {
        COMP (THIS, Authors.Name(QueryTrack2Authors(1))),
        QUERY QueryTrack2Authors(T)
    }
    ...
}
```

```
NU Authors.Name =
{
    ROOT( A )
    ATTRIBUTE name = ...
}
```

```
QUERY QueryTrack2Authors( dm:Track T ) =
{
    SELECT { A }
    FROM { {T}<dm:accepted>{}<dm:writtenBy>{A} }
}
```

B. Links

Navigation units represent a modular specification means of structuring and retrieving presented information for single navigation nodes (e.g. web pages). Links in Hera define navigation transitions between different NU and allow composing a whole web application from NU.

Definition 3.15 (Link) Link is a tuple \( link = <N,a> \) where \( N \subseteq NU \times NU \) is a navigation NU relationships and \( a \) is the anchoring constant expression specifying a (constant) information presented by the link.

The concept of “link” is similar to the concept of “composition”. The main difference is the different semantics of them. Whereas a composition defines a relationship between
NU whose instances are displayed at the same time, a link defines a relationship between
NU where the destination NU is instantiated and displayed after some user interaction
(clicking on the link anchor). Similarly as with data compositions links can also restrict
the data content of the target unit.

**Definition 3.16 (Data Link)** A data link is a tuple \( dlink = < N, q > \), where:

- \( N \subseteq NU \times NU \) is a navigation relationship and
- \( q \) is a parameterized retrieval link query. This is a regular retrieval query that in
  addition defines the anchoring expression. If \( N(nu_S, nu_D) \) then \( q \) defines the extent
  (data instances) of root(\( nu_D \)) and the input data set of \( q \) is the extent of root(\( nu_S \)).
  The query \( q \) also defines the information presented as anchor.

Selecting links is triggered by clicking on an active anchor displaying a value of a
root(\( u_D \)) instance.

**Example 3.9 (Data Link)** The following link example defines a navigation relationship
between the Track.Details NU presenting a track’s information and the Author.List NU
presenting information about authors. The link uses the QueryTrack2Authors query that
is a query similar to the one from Example 3.8 extended with ANCHOR clauses.

\[
\text{NU Track.Details(dm:Track AS T)} = \\
\{ \\
\text{ROOT} \{ T \} \\
\ldots \\
\text{DLINK Track2Authors} = \\
\{ \\
\text{NAVIGATION} (\text{THIS}, \text{Authors.Details(QueryTrack2Authors(1))}), \\
\text{ANCHOR} \text{AnchorAuthor}, \\
\text{LQUERY} \text{QueryTrack2Authors(T)} \\
\} \\
\text{NU Authors.Details(dm:Author AS A)} = \\
\{ \\
\text{ROOT} \{ A \} \\
\text{ATTRIBUTES} \\
\ldots \\
\} \\
\text{LQUERY QueryTrack2Authors( dm:Track AS Track )} = \\
\{ \\
\text{SELECT} \{ A \} \\
\text{FROM} \{ \{\text{Track}\}<dm:accepted>{}\<dm:writtenBy>{\{A\}} \} \\
\text{ANCHOR} \{ \{A\}<dm:firstName> + " " + \{A\}<dm:lastName> \} \\
\} \]
A data link specification uses an extended query that specifies information that a concrete link instance presents. This information is typically related to the navigation target instance. The ANCHOR clause specifies the displayed values. This clause is an extension of the projection defined in the SELECT clause.

C. Attributes

The content of NU is defined by existing compositions and links. Due to the way NU are defined they represent only data restrictions and composition of child units. Since the main purpose of NU is to specify how to present information there is a need to define facilities characterizing the presentation of concrete data values. Attributes are expressions that determine what literal values of a input concept are displayed and how. Attribute expressions can also use arithmetic and string operators. Attributes are defined within NU and they are only means of information presentation specification. The input concept for all attributes in a single NU is the root concept.

Definition 3.17 (NU Attribute) NU Attribute Attr are expressions based on literal path expressions (path expressions ended by literals, constants or arithmetic/string operators listed in Table 3.1.

Attributes are expressions based on a parameter (an instance of a required concept). Note that they can also be used as a child NU in compositions.

Example 3.10 (NU Attributes) The following unit Author.Details specifies a number of attributes by using path expressions based on the dm:Author root concept that is input as a parameter to this unit, using constants and using arithmetic and string operators. Note the COUNT operator that is applied to a concrete instance of dm:Author in a concrete unit instance.

NU: Authors.Details =
{
    ROOT( dm:Author A ) =
    {
        SELECT { A }
    }

    ATTRIBUTE name = {"Name: " + {A}<dm:firstName> + " " + {A}<dm:lastName> }
    ATTRIBUTE affiliation = {"Affiliation: " {A}<dm:affiliation> }
    ATTRIBUTE nuPapers =
    {
        "Number of papers: " +
        STRING(COUNT({A}<dm:wrote>{<dm:title>}))
    }
}
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a+b</td>
<td>Addition for numerical types, concatenation for string types</td>
</tr>
<tr>
<td>a-b</td>
<td>Subtraction, applicable only to numerical types</td>
</tr>
<tr>
<td>a*b</td>
<td>Multiplication, applicable only to numerical types</td>
</tr>
<tr>
<td>a/b</td>
<td>Division, applicable only to numerical types</td>
</tr>
<tr>
<td>sum(a)</td>
<td>Sum over all instance values, applicable only to numerical types</td>
</tr>
<tr>
<td>min(a)</td>
<td>Minimum from all instance values, applicable only to numerical types</td>
</tr>
<tr>
<td>max(a)</td>
<td>Maximum over all instance values, applicable only to numerical types</td>
</tr>
</tbody>
</table>

Table 3.1: Operators that can be used in attribute expressions

Attributes can be understood as being atomic NU that cannot contain links, compositions or other attributes. In addition, there are a few other notable types of NU useful for the design of web applications:

- **Context-free NU** (CFNU) are NU without the root concepts and whose content contains no or only constant attributes, no or only non-data links and no or only non-data compositions. It is only a container that does not put any constraints on its content.
- **Menus** are CFNU that contain only the non-data links.
- **Top-level NU** are NU that do not appear in any composition or data composition as child units in the same application model.

**Example 3.11 (Menu)** The following script demonstrates the shortcut syntax for defining menus (they can be specified also as regular NU):

```plaintext
MENU: ConferenceMenu =
{
    "Conference tracks", am:Track.List,
    "Accepted papers", Paper.Accepted,
    "Members of PC", PCMember.List
}
```

*Instead of full link descriptions, only constant anchors and names of target NU are specified.*

### 3.2.3 Application Model

An AM consists of elements described in previous text. This section summarizes the definition of AM and explains how the elements of AM are instantiated. In order to do so a notion of navigation context describing a state of navigation session is introduced.
3.2 Basic Application Modeling

A. AM Definition

Definition 3.18 (Application Model) An application model is a tuple $\textbf{AM} = \langle n_0, N, \lambda \rangle$ where:

- $n_0$ is the starting navigation unit,
- $N$ is the set of all navigation units in $\textbf{AM}$ (including $n_0$); all $n \in N$ must have an association with at least one different unit by means of links or compositions,
- $\lambda : N \cup A \cup C \cup L \rightarrow \text{string}$ is a naming function that assigns a name to all NU and their components (attributes, compositions and links).

Example 3.12 (Application Model) Consider the sample domain (conference submission and review system) and the detailed example presented later in Paragraph D. The AM contains six NU and it is:

\[
\begin{align*}
\ n_0 &= \{\text{Conference.Main}\} \\
\ N &= \{\text{Conference.Main, Tracks, Track.Detais, PC, Paper.Info, Paper.Detais}\} \\
\ \lambda &= \{(\text{Conference.Main, ”Conference.Main”}), (\text{Tracks, ”Tracks”}), \ldots \}
\end{align*}
\]

Note that the naming function allows using different model identifiers and names that do not necessarily have to be the same. Additionally, in accordance with Definition 3.12, the naming function also maps identifiers of all unit subcomponents.

B. AM Instantiation

The context of navigation (further only context or NC) describes the current state of navigation within a single navigation session of a single user. Navigation itself is described by the term navigation space. The notions of navigation context and navigation space help to explain the instantiation of AM.

Definition 3.19 (Navigation Context) A navigation context is a tuple $\textbf{C} = \langle n, R \rangle$, where:

- $n$ is the currently visited NU (its class),
- $R$ is the instance of the root concept of $n$.

Let us consider a navigation space as the set of all possible navigation contexts reachable by following web pages, constructed using a concrete application model and a concrete domain, with a domain vocabulary of which terms are (only) used in the application model. Then we can define it as a simple transition system. Instantiation is then a process of populating concrete AM navigation units happening with all transitions.
Definition 3.20 (Navigation Space) Let us consider a domain $\text{Dom}$, its domain model $dm$, an AM based on the domain model $am$. A navigation space $N$ is a transition system described by the tuple $< C_{\text{nav}}, c_0, \rightarrow >$ where:

- $C_{\text{nav}}$ is a set of all reachable navigation contexts using domain $\text{Dom}$ and the models $dm$ and $am$,
- $c_0 \in C_{\text{nav}}$ is the starting navigation context and
- $\rightarrow : C_{\text{nav}} \times C_{\text{nav}}$ is a transition between contexts. This transition is triggered by following links.

Definition 3.21 (AM Instantiation) Let us consider a navigation space $N$ associated with a domain $\text{Dom}$ and models $dm$ and $am$. Instantiation of the model $am$ is the construction of page instances corresponding to contexts $c \in C_{\text{nav}}$. For the $c = < n, R >$ the top-level NU is instantiated as follows:

- The root query of $n$ is evaluated (if it exists) using $R$ as a parameter.
- All compositions of $n$ are evaluated as follows:
  
  - Composition query is evaluated (typically) using the value $R$ as a parameter.
  
  - The child NU is instantiated using this procedure and the new temporary navigation context $c_C = < n_C, R_C >$ where $n_C$ is the child NU and $R_C$ is the instance of the root concept of $n_C$ calculated by the composition query.

- Links are instantiated by evaluation of the link query and by calculating the values of anchoring expressions. Anchoring expressions are transformed into the target format (e.g. HTML)

- Attributes are instantiated by evaluation of attribute expressions. Subsequently they are transformed into the target format.

A navigation space is an abstraction of all possible user navigation paths. AM instantiation is the process of creation of the AM top-level navigation units (corresponding to hypertext presentation of pages) for all navigation contexts. This instantiation can be performed at once for all navigation contexts, or a new page can be constructed for every transition when the transition takes place. We prefer the second approach that becomes necessary when the AM is extended with data content manipulation.
C. Graphical Notation

The Hera AM has a graphical notation that allows a good overview of its structure. This notation has been adopted from the RMM method (see Chapter 2) and uses slice shapes for the representation of navigation units. Although this notation is precise and logical, it is not commonly used and understood and its layout is inefficient for large navigation units (slice shapes are not suitable to contain many attributes and sub-units). For this reason we have refined the notation that now uses rectangular shapes and that is shown in Tables 3.2 and 3.3.

D. Example

An example AM is shown in Figure 3.4. This simple application presents information about conference tracks and PC members. The menu items navigate to Tracks and PC (which are Context-free NU), the first with an index of all tracks represented by the link AllTracks that is not connected to the parent unit. An example attribute that presents a set of values is the authors attribute evaluated within the data composition paper2author:

\[\text{NU Paper.Info(dm:Paper AS P) =} \]
\[
\text{\{ ROOT \{ P \} \}}
\]
\[
\text{\ldots DCOMP paper2author \}
\]
\[
\text{\{ COMP(This, Author.Details(QueryPaper2Authors(1))),}
\]
\[
\text{QUERY QueryPaper2Authors(P))}
\]
\[
\}
\]
\[
\ldots \]

\[\text{QUERY QueryPaper2Authors(dm:Paper P) =} \]
\[
\text{\{ SELECT}
\]
\[
\text{\{ A \}}
\]
\[
\text{FROM}
\]
\[
\text{\{ P<dm:writtenBy>{A} \}}
\]
\[
\}
\]
\[
\ldots \]

This example presents a link associated with its parent unit is the lPaperToPaper link. This link uses directly the information related to the current instance of dm:Paper and
<table>
<thead>
<tr>
<th>Diagram</th>
<th>Counterpart in Hera (Static)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="" /></td>
<td><img src="image2" alt="" /></td>
</tr>
<tr>
<td>Navigation Unit</td>
<td>Slice</td>
</tr>
<tr>
<td><img src="image3" alt="" /></td>
<td><img src="image4" alt="" /></td>
</tr>
<tr>
<td>Context-free Navigation Unit</td>
<td>Constant slice</td>
</tr>
<tr>
<td><img src="image5" alt="" /></td>
<td><img src="image6" alt="" /></td>
</tr>
<tr>
<td>Attributes</td>
<td>Attributes and Connection Property</td>
</tr>
<tr>
<td><img src="image7" alt="" /></td>
<td><img src="image8" alt="" /></td>
</tr>
<tr>
<td>Connecting Properties</td>
<td>Attributes and Connection Property</td>
</tr>
<tr>
<td><img src="image9" alt="" /></td>
<td><img src="image10" alt="" /></td>
</tr>
<tr>
<td>Link</td>
<td>Link</td>
</tr>
</tbody>
</table>

Table 3.2: Graphical notation for AM
Table 3.3: Graphical notation for AM (continued)

passes it as the link parameter to the destination NU. The destination NU uses this parameter usually for its instantiation — for the execution of its root query. The complete Paper.Info unit is:

\[
\text{NU Paper.Info(dm:Paper AS P) = }
\]

\[
\{ 
\text{ROOT } \{ P \} 
\}
\]

\[
\text{DCOMP paper2author = }
\]

\[
\{ 
\text{...}
\}
\]

\[
\text{DLINK lpaper2paper = }
\]

\[
\{ 
\text{NAVIGATION( THIS, Paper.Details(P) ) }
\{ 
\text{LQUERY LQueryPaper2Paper(P) = }
\}
\]

\[
\{ 
\text{SELECT } \{ P1 \}
\text{FROM } \{ P \}
\text{ANCHOR } \{ \text{"Title: " + } P \text{<dm:title>}
\}
\}
\]

\[
\text{...}
\}
\]

Note that link queries are used here in two different ways. The \text{LQueryPaper2Authors} is defined externally and its retrieved values (authors) are referred to by order number of selection expression (we assume it is an author instance) as \text{LQueryPaper2Authors(1)}. The second query \text{LQueryPaper2Paper} is defined within the scope of the link.
Figure 3.4: A graphical representation of a simple navigation view over the conference submission domain

3.3 Conclusion

This chapter introduced the data domain model and the basic application model. The definitions were formalized and accompanied by examples that support the explanation. This chapter addressed Research Topic 2 from Chapter 1 regarding the specification of navigation unit structures and retrieval structures allowing convenient modeling of navigation views on domain data. This specification included navigation units and their composition and link relationships. All navigation units have either a clearly defined mapping to the domain structure, or they can be context-free (not dependent on domain concepts). This basic level AM can serve for building simple web sites offering a web interface for presenting domain data. However, designing complex WIS requires more than that. The main purpose of the basic AM is to serve as the foundation for an extension discussed in the following chapter.
Chapter 4

Advanced Application Modeling

The basic application model explained in Chapter 3 facilitates the specification of static navigation views on domain data. This model contains means for defining data retrieval and presentation on the Web. However, requirements on modern WIS ask for techniques allowing the specification of more advanced functionality, for instance gathering information from users and its processing, dynamic updating of the navigation structure and data content, communication with external systems, etc. Design models (mainly the application model) used for WIS should be able to cope with these requirements. This chapter addresses Research Topic 3 stated in Chapter 1 by presenting an extension of the basic Hera application modeling to satisfy this requirement. This extension contains modeling primitives for the specification of a navigation interface that improves interaction with users and modeling primitives for the specification of the effects of this user interaction. Accordingly, the extended (advanced) AM is divided into a navigation model and a content manipulation model. The extended AM in addition supports communication with external systems using the Web. This chapter explains not only what the extended set of AM building blocks is, but also how they are expressed in the RDFS AM vocabulary.

Section 4.1 motivates the need of the advanced AM and explains its basic structure. The structure and RDFS representation of the basic AM is presented by Section 4.2 as a starting point for the extension. The navigation part of the advanced AM is explained in Section 4.3 where all the extensions and their AM vocabulary representation are discussed. The semantics and representation of the content manipulation model added to AM is explained in Section 4.4. Section 4.5 introduces the advanced AM definition, explains its instantiation and presents a larger AM example. The chapter is concluded in Section 4.6.

4.1 Overview of the Advanced AM

The main requirement for advanced application modeling is the ability to express the design of data-intensive Web applications with advanced functionality that allows not only to present information on the Web but also allows to actively interact with users (or external systems) and to process their feedback at runtime. This general idea is captured
by the following more detailed requirements:

- **Requirement 1**: the ability to express a navigation view constructed at runtime on demand from the domain data. This guarantees that only up-to-date content is presented.

- **Requirement 2**: the ability to express dynamic runtime adaptation of the navigation structure based on the current state of the data content. This is a vital requirement for (but not only for) personalization and context adaptation purposes.

- **Requirement 3**: the ability to express content manipulation that depends on user interaction. This is vital for maintaining (persistent) data state for information systems in general.

- **Requirement 4**: decoupling of the content manipulation specification from the interface specification. This separation facilitates the design and maintenance of WIS.

The extended application model is divided into the navigation model (NM) and the content manipulation model (CMM). NM specifies the navigation structure (perceived as web interface) and CMM determines the effects of events possibly triggered by user actions on the data content. The connection between NM and CMM and between their instances is provided by events originating in the Web interface. CMM is implemented as a set of rules that contain specification of actions allowing to manipulate data content, to invoke functionality of external (Web) systems, and to initialize timers generating time events. An abstract structure of AM is shown in Figure 4.1.


4.2 Extending the Basic AM Structure

In order to specify the advanced features of WIS the existing AM has been extended with modeling elements facilitating the specification of navigation structure and supporting the specification of business logic. Using an event mechanism is motivated by the effort to decouple the specification of the system behavior from the specification of the user interface. The system behavior is expressed using Event-Condition-Action (ECA) rules.

Before explaining the AM extension let us summarize the basic AM elements defined in Chapter 3 and explain their RDFS representation. The elements are organized in a hierarchical structure of meta-concepts used in every concrete AM. Figure 4.2 shows the basic AM vocabulary. The core meta-class is NU (navigation unit). It is a navigation view on a domain concept grouping attributes of its root concept and related concepts. NU can be aggregated by means of Composition subclasses and can be connected with links by means of Link subclasses. Attributes model a special kind of NU that present data values of domain attributes. As explained in the previous chapter, attributes do not contain links or child units. Query is a meta-concept representing all data retrieval queries discussed in Chapter 3. It consists of Projection, Selection and Order parts containing graph patterns, restricting condition and eventually ordering expressions stored as strings. The basic AM offers a versatile specification means that allows to model the navigation interface of a domain. This model has been proposed with Requirement 1 from the previous section in mind.

Figure 4.2: The basic navigation meta-model expressed as RDFS vocabulary
Example 4.1 (Basic AM example) Figure 4.3 shows how a concrete snapshot of AM is represented in terms of subclasses of basic AM vocabulary terms. The initial navigation unit (Conference.main) presents the conference name and a set of links to the track detail NU (Track.Main) displaying all information about a concrete conference track. Conference.main, cname and Track.Main are subclasses of the NU AM vocabulary concept. The cname attribute is associated with its parent Conference.Main by means of Composition_ID1. The link conference2tracks has a link query Query_ID1 that determines the relationship between dm:Conference and dm:Track (dm:hasTracks property) and that retrieves appropriate track information based on user link selection. Note that the NU Conference.Main does not have any restriction (it is optional). An excerpt of the abstract syntax expression for this snapshot is:

\[
\text{NU Conference.Main} =
\{
\text{ROOT} =
\{
\text{SELECT} \{ C \}
\text{FROM} \{ \{C\}<\text{rdf:type}>{dm:Conference} \}
\}
\text{ATTRIBUTE} \text{cname} = \{ "Name: " + \{C\}<\text{dm:cname} \}
\text{LINK} \text{conf2Tracks}
(\text{NAVIGATION(THIS, Track.Name)}, \text{hasTracks}(C))
\}
\]
4.3 Navigation Model in the Advanced AM

The advanced AM consists of a navigation model (NM) that is an abstract specification of the application’s Web interface and of a context manipulation model (CMM) that is the specification of the effects of user actions (and other system events) on the data content. In this section we study NM. We treat CMM in the next section.

4.3.1 Motivation

The basic AM offers a good platform for building a Web interface for the domain data. However, as formulated by Requirement 2 in Section 4.1, the navigation structure can be influenced by the data state stored in the data content. This dynamic adaptation of the navigation structure does not only mean the presentation of fresh data content but also the runtime selection of presentation information, runtime adaptation of possible navigation paths etc. The content manipulation model addressing Requirement 3 and discussed in the following section relies to a large extent on the ability of WIS to capture user input. User input collection is also part of NM and it is contained in the described NM extension presented in Figure 4.4. In order to satisfy the requirements the following modeling elements have been added to the AM-NM:

- **Forms** that allow gathering predefined types of information from users,
- **Inclusion conditions** that allow dynamic (at runtime) inclusion or exclusion of child NU or links and
- **Optional navigation branches** that allow runtime branching of navigation paths.

4.3.2 Forms

Forms can present information but they also allow entering of numerical and textual information and choosing from multiple predefined options. Forms cover a part of Requirement 3 targeting also user input. A form is a collection of user editable input fields of different types that may have predefined (initialized) values that are either constant or retrieved from the data content. After entering or editing the input values a form can be submitted by activating a trigger (displayed as a button) that determines the way the form is processed. A single form can have possibly multiple submission buttons (triggers) associated with different ways of form processing. An extension shown in Figure 4.4 specifies the structure of forms, inputs, inclusion conditions and their relationships. The Form meta-class is a subclass of the NU meta-class and represents a form with a set of input fields represented by the Input meta-class. Inputs are attributes — they can show some initial value (or a set of values for selection inputs) that are determined by the attribute expression. The type of resource (concept) returned by an input is given by the attribute expression of its Attribute superclass. The inputs also define the way in which the user
enters values. This divides inputs into the following categories (classes) that are currently used:

- **Select1outN** input allows users to pick one value from a pre-selected set.
- **SelectNoutN** input allows users to pick more values from a pre-selected set.
- **InputText** input allows users to enter any text (or number).

Forms can be included in parent NU by using Composition subclasses. A form submission triggers an associated event taking the form input values as its argument. The triggering is modeled by the Trigger class that next to a Submission event has an associated automatic link (a subclass of Link with empty anchor that is triggered by an event different from LinkClick). The instance of the link destination NU is determined by the link query but it can also be influenced by data update rules triggered by the form submission event (optionally) before the navigation to the destination NU takes place.

**Definition 4.1 (Input)** An input $I$ combines an attribute that can be changed by users with the specific type of information it returns. It is thus a tuple $I = < a, t >$, where $a$ is the attribute expression and $t$ is the type of the returned information.

**Definition 4.2 (Form)** A form is a NU defined by a tuple $F = < r, q, I, T >$, where:

- $r$ is a root concept,
- $q$ is a root query,
- $I$ is a set of inputs,
- $T$ is a set of triggers defining form processing.
Definition 4.3 (Trigger) A trigger is a tuple \(<l, e, \text{cap}>\) associating an event \(e\) with a link \(l\). The event \(e\) is thrown prior to the processing of the link \(l\). The caption \(\text{cap}\) is displayed within the parent form and it explains (typically very briefly, as a button label) to the user the processing that the trigger will start.

Forms contain the specification of their content (inputs), and the specification of the triggers associated with submission events. Although the processing of forms is specified in CMM the association of triggers and submission events is defined in NM.

4.3.3 Inclusion Conditions

Inclusion conditions allow runtime inclusion or exclusion of nested NU. In this way runtime adaptation of navigation structures, addressed in Requirement 2 in Section 4.1, is implemented. This technique supports dynamic adaptation of the NU structure based on the current data state and user behavior. Inclusion conditions are commonly used in all types of applications but they are vital for the design of adaptive and/or personalized applications. They are defined as Boolean expressions \((\text{BExpression})\) determining if a NU or a link is included into a parent NU. If a condition is evaluated as \(true\) the appropriate element is constructed and rendered, otherwise it is not. The expressions can contain Boolean and relational operators, constants and variables bound in queries (then the queries are part of an \(\text{BExpression}\) instance). Inclusion conditions can be applied to any NU, links or attributes.

Definition 4.4 (Inclusion Condition) An inclusion condition is a tuple \(<q, c>\) where \(q\) is a (optional) retrieval query and \(c\) is a Boolean expression using bound variables from \(q\). An inclusion condition is attached to a NU, (data)link, or attribute and its value determines its inclusion in the instantiated NU.

Example 4.2 (Form and Inclusion Condition) An example of a fragment of AM using a form is given in Figure 4.5. The left part shows an excerpt of an AM with a form for selecting a number of papers to be accepted and a target NU (list of accepted papers), and the right part shows an excerpt of the model as an RDFS graph. The Track.Selection unit specified in this example allows to select multiple submissions for a single track. When the form is submitted the selected papers are displayed in a new page. Note there is no other form processing specified in this example. The Track.Selection NU contains the Paper.Selection form with the input title (subclass of SelectNoutN). The input selection attribute is TitleAttribute (not visible in the AM graphical notation). The navigation destination of the form is determined by the destination property of the form and it is the Track.Accepted NU. The instances of the destination NU are determined by the data state after all rules triggered on the event PaperSelected are executed. As it is explained later in the text an additional group of rules can be defined that is executed just prior the destination NU is constructed but after the form processing rules are executed. An excerpt of the abstract syntax notation describing the Track.Selection unit is:
Figure 4.5: An example of a form and an inclusion condition presented in the Hera AM notation (left), and as RDFS graph (right)

```
NU Track.Selection( dm:Track AS T ) =
{
   ROOT { T }
   IF ( #PC > 0 )
   {
      QUERY
      {
         SELECT { PC }
         FROM
         {
            {session:user}<rdf:id>{PC}
         }
         WHERE
         {
            {PC}<rdf:type>{dm:PCMember}
         }
      }
      SHOW Papers.Selected(T)
   }
}
FORM Papers.Selected( dm:Track AS T ) =
{
   ROOT( T )
   InputNfromN Paper.Input =
```
4.3.4 Conditional Navigation

Concrete application cases show the need of modeling optional navigation paths that are decided upon at runtime. Situations where following the same link possibly leads to different destination units depending on the current application data state are not exceptional. A typical example is handling undesired states caused by database systems or external systems. For instance, if a database transaction is rolled back within the processing of a submitted form it is convenient to redirect the original navigation path to an error path that handles the exception, because it may not possible to follow the original course of action (e.g. when the original navigation path explicitly depends on successful execution of data updates, but these failed). Conditional navigation specification — addressing Requirement 2 in Section 4.1 — uses automatic links having multiple possible destination units, where every destination unit has an associated condition that is evaluated when the link is activated and that decides which path is to be followed. Automatic links specify enforced navigation that happens after a rule is executed. They are used for navigation caused by the processing of forms. A link with conditional branches is always “automatic”, because it does not make sense to use navigation conditions if a user selects a concrete link (for adaptation purposes an inclusion condition can be used). The following definition extends the notion of “automatic link” to links with multiple destinations.

**Definition 4.5 (Autolink)** An autolink is a tuple \(< s, C, D, Q \rangle\), where:

- \(s\) is the source NU,
- \(C\) is a set of \(k\) mutually exclusive expressions \(c_1 \ldots c_k\) the disjunction of which is always true,
- \(D\) is a set of \(k\) destination NU \(d_1 \ldots d_k\),
- \(Q\) is a set of \(k\) queries \(q_1 \ldots q_k\) with the data projections \(\pi(q_i) = \text{root}(d_i)\) all bound with \(\text{root}(s)\).

Conditions attached to an autolink determine which navigation path is taken and the (optional) queries retrieve values of link parameters that are used in the destination units (e.g. restriction of data content of the destination unit).
Example 4.3 (Conditional Navigation) An example of the navigation condition handling a possible database error by showing an error message is shown in Figure 4.6. It is the same example as the previous one, but here the autolink providing conditional navigation is shown in detail. An excerpt of the AM is represented in the Hera graphical notation and as an RDFS graph (only the part modeling navigation condition is shown). Conditions are evaluated at runtime and in the case of a database error an instance of ErrorPage is rendered. Information about the result of a data manipulation query is stored in a context data space during the form processing. The abstract syntax notation is:

FORM: Papers.Selected( dm:Track AS T ) =
{
  ...
  TRIGGER select = {
    AUTOLINK PapersSelected =
    {
      NU THIS,
      PATHS
      {
        PATH {
          QUERY
          {
            SELECT { P }
            FROM { {T}<dm:accepted>{P} }
          }
          IF ( STATUS = 0 )
          {
            QUERY = StatusQuery( ) : STATUS
            NAVIGATE Track.Accepted(P)
          }
        }
        PATH {
          QUERY
          {
            SELECT { P }
            FROM { {T}<dm:accepted>{P} }
          }
          IF ( STATUS = 1 )
          {
            QUERY = StatusQuery( ) : STATUS
            NAVIGATE ErrorPage
          }
        }
      },
      EVENT papersSelected(Paper.Input.Selected),
      CAPTION "Select"
  }
}
Figure 4.6: An example of conditional navigation presented using the Hera AM notation (left), and RDFS (right)

```sql
QUERY StatusQuery() : STATUS =
{
    SELECT { STATUS }
    FROM { {session:session}
            <session:lasttransactionstatus>{STATUS} }
}
```

Note that the queries of a autolink corresponding to different navigation paths are always evaluated after the form processing (triggered by the event `EventPapersSelected` and defined in CMM)

### 4.3.5 Inheritance in NM

Concept and property inheritance are mechanisms where derived (specialized) concepts inherit all relationships and properties of the parent concepts. They are a variation of the transitive `isA` property. `SubClassOf` and `subPropertyOf` are `isA` relationships of RDFS. These special properties can be applied to potentially all building elements of AM. However, in order to facilitate the application design and keep it transparent, the application model in Hera restricts the inheritance to only navigation units. Possible inheritance of CMM elements such as rules — explained later in this dissertation — could bring ambiguity and non-transparent behavior. Moreover, rules do not represent domain or application objects but only recipes for updating the data content.
Definition 4.6 (NU Inheritance) Let consider two navigation units based on the same domain model \( u_0 = \langle r_0, q_0, C_0, L_0, A_0 \rangle \) and \( u_1 = \langle r_1, q_1, C_1, L_1, A_1 \rangle \). In order for \( u_1 \) to be inherited from \( u_0 \) the following properties must be satisfied:

- \( r_1 \preceq r_0 \) if \( u_0 \) has a root concept (in not context-free),
- \( C_0 \subseteq C_1 \),
- \( L_0 \subseteq L_1 \),
- \( A_0 \subseteq A_1 \).

Figure 4.7: An example of the NU specialization

An example of inherited units is shown in Figure 4.7. The NU Person.Main has two specializations Reviewer.Main and PCMember.Main. The first one represents information relevant for reviewers and the second one information relevant for PC members. Which concrete specialization is used is decided at runtime and it depends on the exact type of the root instance (it is either Reviewer or PCMember). This NU specialization is correct since the domain concepts Reviewer and PCMember are sub-concepts of Person. This example demonstrates the benefits of this kind of inheritance for personalization purposes.

4.4 Content Manipulation Model

In the previous section we discussed navigation model elements. In this section we explain the extension of AM that allows to specify data content manipulation that is essential for the development of advanced WIS.
4.4.1 Motivation

In order to define advanced WIS features and in line with Requirement 3 in Section 4.1 we need to model (persistent or non-persistent) updates of the data content possibly triggered at runtime. Effects of these updates are reflected also in information presented by the Web interface. Although the information updates can be directly defined in AM it is convenient to separate this specification from the navigation specification (Requirement 4 in Section 4.1). There are multiple reasons for it; one of the most important is the separation of concerns — possible changes in the navigation structure of the application do not affect the specification of WIS data transactions. In other words, business logic of WIS (and its specification in CMM) is less dependent on the presentation logic (and its specification in NM). In summary the following concepts are added to AM:

- Events triggering different types of actions including data updates and
- Actions that can be triggered by events providing:
  - Update operations,
  - Invoking external functions and
  - Triggering timed (based on time) events.

Data update operations can be triggered by events caused by users or by the environment. Event-Condition-Action (ECA) rules, originally introduced for active databases, appear to be a natural and convenient choice for modeling event effects. An ECA rule has the form $e[c] \rightarrow a$ where the action $a$ is performed when the event $e$ occurs but only if the condition (guard) $c$ is satisfied.

4.4.2 Events

Events are signals, possibly triggering actions associated with the occurrence of user interaction, transitions in navigation space, passing specified time intervals, entering exceptional system states, etc. Events compose an interface layer separating the navigation structure specification from the content manipulation specification. In the context of Hera AM we define events as follows:

**Definition 4.7 (Event)** Event is a tuple $< n, \tau, d >$ where $n$ is the event identifier (name), $\tau$ is the event type, and $d$ is a parameter characterizing the context of the event occurrence. Events are divided into basic types characterizing their sources:

- navigation transitions caused by users or the system,
- passing specified time intervals,
- entering exceptional states of the system,
- external causes (external calls)
Events may carry parameters initialized at the moment of their triggering. It is important to realize the distinction between the event as a modeling concept (Definition 4.7) and an event instance that is constructed at runtime. An event instance occurs at a specific time and it carries parameters initialized when the event instance is created.

A. Navigation Events

Navigation events are triggered by the Web interface and they are (almost always) caused by user actions. Navigation events are associated with navigation transitions and their specification contains a set of items that are also contained in the event parameter set:

- an event identifier used for the specification of CMM rules,
- a reference to a Web interface element causing the event, that can be a link or a form trigger,
- a reference to the destination NU reached by the navigation transition,
- the type of the event specifying its order in the process of NU instantiation (before or after) order and
- optional non-default parameters that can be transferred by an event instance to CMM rules the event triggers.

Every navigation event is associated with so called destination NU and every navigation event instance is associated with an instance(s) of the destination NU root concept. The destination NU and the root concept instance is determined by the Web interface element causing the event. Navigation events can be associated with more than one element (for instance they can use their information content), but exactly one element must be the triggering element that can be a link or a form trigger. Navigation events are divided into the following types suggesting the event ordering related to their destination NU instantiation:

- BeforeLoad events occur together with navigation transitions but they are fired before the destination NU is instantiated. This type of events is essential for situations when a data update operation needs to be performed before the destination page is shown (the destination page might require updated data state).

- AfterLoad events occur after the navigation transition is finished (after the destination NU is instantiated). This type of events is used for triggering actions that must (or may) be executed after the destination page is displayed (e.g. measuring time of viewing the page).

A default parameter of a navigation event is the aforementioned instance of the destination NU root concept. This instance is decided upon at runtime, for example by selecting a link instance by the user or by evaluating a query associated with a form trigger. For possible additional parameters data initialization queries providing values of the parameters must be also specified (for all types of events).
Example 4.4 (Navigation Event 1) Consider the link conf2tracks from Example 4.1. Figure 4.8 shows a simple specification of the event c2tClicked without additional parameters. The only parameter is the default parameter (the destination root concept) initialized automatically. Note that because there are no additional parameters the declaration of the event in this manner is not necessary. The event reference can be directly used in the CMM.

\[
\text{EVENT c2tClicked = Navigation( conf2tracks ) AfterLoad}
\]

Figure 4.8: Representation of a “follow link” event declaration using abstract syntax (left) and AM vocabulary (right).

Example 4.5 (Navigation Event 2) Figure 4.9 shows the specification of a navigation event caused by form submission through the form trigger Process (referred to as F.Process using the form alias). In this case besides the default parameter (the completed form instance) a parameter containing selected papers is used. It allows a simple way of transferring information required in an event handler (a rule in the CMM). Query_ID1 defines how the parameter value is retrieved.

B. Timed Events

In WIS often certain operations must be triggered at, or after a specified time, or at regular time intervals. Consider important points of time defined within a conference submission WIS, like for example conference deadlines that may trigger PC member notifications, changes of the Web interface for authors, etc. In such situations timed events appear to be a useful triggering mechanism. Timed events are fired by timers discussed in Section 4.4.4. Events of this type can be triggered once or periodically depending on the concrete situation. Default parameters of timed events are:

- an \textit{event identifier} used for the specification of CMM rules,
- the \textit{time of occurrence} or time stamp and
- the \textit{periodicity}.

Example 4.6 (Timed Event) Figure 4.10 defines a timed event. For this example there must be a timer with the same name as the event name: TEvent24per.
EVENT papersSelected = 
{
    Navigation( am:Papers.Selected AS F,
        F.Process )
    BeforeLoad
    {
        PARAMETER
        {
            QUERY
            {
                SELECT { P }
                FROM
                {
                    {F}<am:input-ref>{am:title AS A},
                    {dm:Paper AS P}<dm:title>{A}
                }
            }
            SET Papers = { P }
        }
    }
}

Figure 4.9: Representation of a submission event declaration using the abstract syntax (left) and AM vocabulary (right).

C. Exception Events

Exceptions are undesired situations occurring in WIS (or information systems in general). There are different types of mechanisms for handling those situations but in our method we consider a light-weight technique that allows to handle exceptional situations and that is fully a responsibility of the designers. For a small predefined set of undesired situations (currently only database exceptions) exception events are generated that can be handled by CMM rules. Such rules can be defined for all user-triggered data processing. The rules can, in case of exceptions, change the presented information (e.g. to display an error message). The specification of an exception event type contains the following:

- an exception event identifier used for triggering exception handling CMM rules,
- an exception type specifying the type of the exception,
- an exception description.

Note that exception types are predefined and their instantiation depends on a concrete type. Exception event instances contain besides the specified information also an error code originating in the exception source (e.g. database error code). The description provides textual explanation of the exception provided by the exception source.
EVENT TEvent24per = Timed()

Figure 4.10: Representation of a timed event declaration using abstract syntax (left) and AM vocabulary (right)

Example 4.7 (Exception Event) The following exception event declaration specifies a database exception event that can occur when executing Query_ID1.

EVENT EDBError = Exception
  (Exceptions.DBException,
   Query_ID1,
   "Database Error"
  )

Figure 4.11: Representation of an exception event declaration using abstract syntax (left) and AM vocabulary (right).

D. External Events

External events allow WIS to react to external systems. In other words they expose a Web interface that can be invoked by another (external) system. Because the Web is the natural WIS environment the preferred communication means is the use of Web Services (WS) for which standard protocols exist. Declarations of external events are transformed into WSDL [Chinnici et al., 2007] specifications that describe a WS API accessible by Web agents (external WIS). For remote functionality invocation and for information exchange WS use SOAP [Gudgin et al., 2007] messages. The specification of an external event contains the following:

- an external event identifier used for triggering CMM rules,
- a specification of input parameters containing their (xsd) types and their names.

The specified parameters allow to transport messages from external systems and they are available as event parameters when an instance of an external event is created.

Example 4.8 (External event) Figure 4.12 shows the specification of a SOAP message reception. It is a service providing a weather forecast for a requested date and place in the USA (can be used by the conference site to provide a weather forecast for the conference
EVENT EExtAuthorsRequest = External
    (xsd:string systemID,
     xsd:string trackName
    )

Figure 4.12: Representation of an external event declaration using the abstract syntax (left) and the AM vocabulary (right).

period). It contains the service URI and the message name. MsgID_ID1 in the RDFS representation contains the message identification data.

The specification of other types of events is straightforward and is similar to the presented examples. The specification of composed events useful for advanced techniques of application design is discussed in Chapter 5. The following section explains actions that are triggered by events and that provide effects on the data content or on the environment.

4.4.3 Conditions

Conditions are an important part of ECA rules. They allow to control the execution of actions based on the state of the system. Conditional expressions in our model can use parameters of the event from the ECA rule, but can also use data values retrieved from entire WIS data content using an associated query. Variables used in such query can further be used in expressions within the action in the same ECA rule. The specification of conditions is similar (in fact equal) to the specification of inclusion conditions. The only difference is the application of the two different types of conditions. We define condition as follows.

Definition 4.8 (Condition) A condition is a tuple $< c, q >$ where $c$ is a conditional expression and $q$ is an optional data retrieval query providing the conditional expression with data values.

Example 4.9 (Condition) The following condition determines whether a reviewer with a given ID (uid) exists. It uses a query for retrieving the number of reviewers with the given ID.

4.4.4 Actions

Actions are effects triggered by events (discussed before) if appropriate guarding expressions are satisfied. Every action can optionally trigger other rules by generating a new event that is declared in the action body, although this mechanism must be used carefully (or avoided) due to possible cyclic chains of events that might cause WIS malfunction.
IF ( COUNT(R) > 0 )
{
  QUERY
  {
    SELECT { R } FROM
    {
      {dm:Reviewer}<rdf:id>{R},
      {R}<dm:cid>{ID}
    }
    WHERE { ID = uid }
  }
  ... action specification ...
}

Figure 4.13: Representation of an exception event declaration using abstract syntax (left) and AM vocabulary (right).

Actions implement the functionality of WIS and they allow updates of the WIS data state, updates of the WIS Web interface exposed to users and eventual WIS communication with external systems (together with the event mechanism). In the context of the Hera method we define actions as follows.

**Definition 4.9 (Action)** An action is a tuple \( <a, \tau, P, e> \) where \( a \) is an optional action identifier (name; it is optional because actions can be defined in rules directly), \( \tau \) is the action type, \( P \) is a set of parameters consumed by the action and \( e \) is an optional event that is triggered when (after) the action is performed. Actions are divided into basic types characterizing their effects:

- content manipulation actions,
- invocation of external functionality and
- starting timers.

**A. Content Manipulation Actions**

Content manipulation actions allow changing the system data state. In the Hera content manipulation specification it is irrelevant whether changes are permanent or volatile (this can be distinguished by the target model). All content manipulations are expressed by content manipulation queries that use graph patterns. The notion of graph patterns as described in Chapter 3 is extended with instance creation and instance destruction operators. These operators used in graph patterns explicitly specify concept (or property) instances that should be created or removed.
All query facilities in Hera are based on the SeRQL [Broekstra and Kampman, 2001] query language. We use an extended version that relies on query preprocessing to generate a resulting set of SeRQL queries and Sesame [Broekstra and Kampman, 2001] API calls. The specification of SeRQL however does not yet contain data manipulation operations (these can be performed using Sesame API calls). The introduced content manipulation operators are the most essential part of the data manipulation queries in Hera. The content manipulation query concept is vital for the extended AM and we therefore define it here. For this purpose we define the discussed operators and extend Definitions 3.8 and 3.9.

**Definition 4.10 (Content Manipulation Operator)** Consider a domain vocabulary \(< C, P, T, A >\). Content manipulation operators are written in the form \(#op(d)\) where \(d\) is a concept or property \(d \in C \cup P\) or a variable representing them. The operators are of two types:

- \(#\text{new}(d)\) is a construction operator. This operator creates and returns a new instance of a concept or property. It generates a unique name for the new resource within the given namespace.
- \(#\text{remove}(d)\) is a removal operator. It removes an instance(s) of the resource given by type \(d\).

Content manipulation operations are intended to be used within graph patterns. This way of use allows us to restrict created (removed) resources to those occurring in data graphs from the data content matching the graph patterns.

**Definition 4.11 (Extended Data Path)** Consider a domain vocabulary \(< C, P, T, A >\). An extended data path is an expression in one of the forms:

\[
\{\text{Cterm}_1, B(c_1)\} \text{Pterm}_1, B(p_1)\{\text{Cterm}_2, B(c_2)\}...\{\text{Cterm}_n, B(c_n)\} \quad (4.4.1)
\]

\[
\{\text{Cterm}_1, B(c_1)\} \text{Pterm}_1, B(p_1)\{\text{Cterm}_2, B(c_2)\}...\{\text{Cterm}_n, B(c_n)\}l_n, B(l_n) \quad (4.4.2)
\]

where \(\text{Cterm}_i\) is:

- \(c_i \in C\) or
- \(#\text{op}(c_i)\) where \(#\text{op}\) is a content manipulation operator

and \(\text{Pterm}_i\) is:

- \(p_i \in P\) or
- \(#\text{op}(p_i)\) where \(#\text{op}\) is a content manipulation operator.
In a path expression only one type of operator can be used. For all concepts and properties in \( \pi \) (including the ones used with operators) holds: \( \forall c_i, p_i, c_{i+1} : c_i \in \text{Dom}(p_i) \land c_{i+1} \in \text{Ran}(p_i) \) and \( l_n \in \text{Attr}(c_n) \). The optional \( B() \) expressions are binding expressions for concepts/properties containing variables bound to a concrete concept or a property.

Extended path expressions can use only construction operators or destruction (removal) operators. Extended graph patterns are built from extended path expressions. Extended graph patterns must also be purely construction ones or destruction ones. We use this constraint to avoid inconsistent and undecidable data states.

**Definition 4.12 (Extended Graph Pattern)** An extended graph pattern is a non-empty set of extended data paths \( \{\pi_i|i = 1\ldots n\} \) that are in the form described in Definition 4.11. All content manipulation operators used in an extended graph pattern must be of the same type.

Content manipulation queries must also use only one type of operator. This divides them into construction queries and removal queries. Update queries are not implemented yet, but they can be partially replaced by multiple construction and removal queries.

**Definition 4.13 (Content Manipulation Query)** A content manipulation query is the tuple \( < \tau, \pi, \sigma, \kappa > \) where:

- \( \tau \) is the type of content manipulation query constraining the type of operators used in \( \pi \):
  - CONSTRUCT queries that allow only using \#new operator and
  - REMOVE queries that allow only using \#remove operator.
- \( \pi \) is the data manipulating projection represented by an extended graph pattern,
- \( \sigma \) is the data selection represented by a graph pattern and
- \( \kappa \) is a binding expression restricting values of binding variables in \( \sigma \).

Although SeRQL does not contain a data manipulation specification it is possible to express the semantics of the defined content manipulation queries by using the semantics of the SeRQL CONSTRUCT query (graph data retrieval query) as follows:

- A CONSTRUCT data manipulation query has the same effect as the inclusion of the result of the analogical CONSTRUCT SeRQL query (having the same projection without operators and selection) in the original data graph.
- A REMOVE data manipulation query has the same effect as removing the result of the analogical CONSTRUCT SeRQL query (having the same projection without operators and selection) from the original data graph.
Note that data graphs contain only data resources (instances) so including or removing data graphs is analogical to adding or subtracting sets of instances. More details about the actual implementation of content manipulation queries can be found in Section 7.1.6

**Example 4.10 (Content Manipulation Query)** Figure 4.14 shows a business rule that changes the state of a single paper to “accepted” by adding the dm:accepted property instance to the existing dm:Paper instance. The trigger of the rule is an event PSel that is associated with a paper selection form trigger. The construction query throws the exception DBError in the case of a database error. The exception event is generated automatically when the operation can not be finished. An exception rule can use the default parameters of the DBError exception and it can be used later in a conditional navigation.)

```plaintext
RULE Insert_example =
{
  ON PSel(S)
  IF {true}
  {
    CONSTRUCT
      {
        {Track}新(dm:accepted)>{P}
      }
    FROM
      {
        {P}<rdf:type>{dm:Paper};
        <dm:title>{T};
        <dm:for>{Track},
        {Track}<dm:tname>{N},
        {S}<form:title>{Input},
      }
    WHERE
      {
        ( N LIKE "Track1" ) AND
        ( T LIKE Input)
      }
    THROWS DBError
  }
}
```

Figure 4.14: Representation of a rule with a data manipulation query using the abstract syntax (left) and AM vocabulary (right).

**B. Invoking External Functions**

External calls allow the communication of the systems with external systems. This mechanism provides the opposite direction of communication compared to external events dis-

...
cussed in Section 4.4.2, Paragraph D. These techniques are often used together. Invoking external functions is realized using WS, namely by using SOAP and WSDL. For the specification of an external function that is publicly accessible as a WS via WSDL interface the following is specified:

- a service name,
- an operation name,
- a specification of output messages (operation parameters) together with (optional) data retrieval query providing them,
- a specification of an input message (return value) in the form of a content manipulation query.

The specification of an external action is transformed at runtime into a SOAP call addressing the specified service and operation. The operation is finished when the return value is accepted and the content manipulation query applies the return value to the current data state.

**Example 4.11 (External call)** Figure 4.15 shows the representation of declaring an external call retrieving ISBN information for a provided ISBN code that returns a publication description.

**C. Initializing Timers**

Timers generate periodic events or single delayed events (as discussed in Section 4.4.2, Paragraph B.) from the moment of their initialization within a rule body. The initialization is performed by the `StartTimer` action that uses parameters determining the event name, its periodicity and a period. The `StopTimer` command with the event (timer) name as a parameter stops a periodic timer. The following is specified for a timer initialization:

- a timer event identifier,
- the periodicity of the timer (periodic or single),
- the tick time (period).

**Example 4.12 (Starting timer)** Figure 4.16 represents the initialization of a periodic timer that triggers the MyTimerEvent event every 24 hours from the moment of performing the timer initialization action.
GetISBN(code) : description =
ExternalCall
{
    "http://.../isbn.asmx?WSDL",
    "GetISBNInformation",
    IN
    {
        code xsd:string,
        QUERY
        {
            SELECT ...
        }
    }
    OUT
    {
        description xsd:string,
        UPDATE
        {
            CONSTRUCT ...
        }
    }
}

Figure 4.15: Representation of an external WS call providing ISBN information represented using the abstract syntax (left) and AM vocabulary (right).

4.4.5 CMM Vocabulary

The examples we used for explaining event, condition and action declarations demonstrated bits and pieces of the RDFS representation of a CMM based on the CMM vocabulary. In this section we put it all together and briefly discuss the overall CMM vocabulary. Figure 4.17 shows a simplified version. The core of the vocabulary is the Rule concept with actions, conditions and events.

Note the hierarchy of events divided into navigation events (the Navigation class) caused by user actions using the Web interface modeled by AM, exceptions caused by a database engine (the Exception class), external events caused by external systems e.g. responses to web service calls (the External class) and timed events caused by previously triggered timers (the Temporal class).

Conditions are defined be expressions stored as strings. Expressions can use values retrieved by optional queries that are part of a condition.

Actions are divided into content manipulation actions (the Update concept), invocations of functionality of external systems via web services (the ExternalCall concept) and timer initialization actions (the Timer concept). Update actions are further divided into construction and removal actions that are based on extended graph patterns. External calls
MyTimerInit = StartTimer
{
    MyTimerEvent,
    PERIODIC,
    24h
}

Figure 4.16: Representation of timer initialization using the abstract syntax (left) and AM vocabulary (right).

define WS, operation and input and output parameters together with their initialization and storage. Timers can be started or stopped and they are periodic or single.

4.5 The Extended AM

4.5.1 AM Re-Definition

In the previous chapter we have defined the basic AM. The advanced AM is the basic AM extended with additional navigation modeling primitives providing structural adaptation of the navigation structure and with a content manipulation model providing runtime updates of the data content. These extensions are not reflected in the AM definition introduced in Chapter 3. There are also differences between the instantiation of an advanced AM and
the instantiation of a basic AM. The advanced AM is defined as follows.

**Definition 4.14 (Application Model)** An application model is a tuple \(< n_0, N, E, R, \lambda >\), where:

- \(n_0\) is a starting navigation unit,
- \(N\) is a set of all navigation units,
- \(E\) is a set of event types that trigger rules in \(R\),
- \(R\) is a set of ECA rules for which holds \(\forall e \in E : (e[g] \rightarrow \text{action}) \in R\) and
- \(\lambda : \{n_0\} \cup N \cup A \cup C \cup L \rightarrow \text{string} \) is a naming function that assigns a name to all NU and their components (attributes, compositions, and links).

### 4.5.2 AM Instantiation

During a WIS operation the AM specifying the WIS is instantiated on demand. This means that a concrete page represented by a top-level navigation unit is populated with data at the moment it is needed, or more concretely when a user follows a link to the page or submits a form that causes the navigation to this NU. The description of the AM instantiation process helps to clarify the purpose and relationships of AM elements and their semantics. In this section the NU instantiation and the rule triggering process is described.

In order to explain the instantiation of the extended AM we also extend the notion of navigation context introduced in the previous chapter with the notion of “last event”.

**Definition 4.15 (Navigation Context)** A navigation context is a tuple \(< n, R, e >\) where:

- \(n\) is the currently visited NU,
- \(R\) is the instance of the root concept of \(n\),
- \(e\) is the last event that occurred within the system.

Instantiation of the extended AM is to a large extent similar to instantiation of the basic AM discussed in the previous chapter. An important difference is the application of CMM rules triggered by navigation events. Due to the asynchronous nature of timed and external events their effects are not captured by the following definition, but their possible effect on the data state is naturally taken into account in the process of AM instantiation. AM instantiation is defined in the context of a concrete domain and navigation model. It uses the notion of navigation space introduced in Definition 3.19 as the set of all possible navigation contexts.
Definition 4.16 (AM Instantiation) Let us consider a navigation space $N$ associated with domain $Dom$ and models $dm$ and $am$. Instantiation of the model $am$ is the construction of page instances corresponding to contexts $c \in C_{\text{nav}}$. For the $c = \langle n, R, e \rangle$ the top-level NU is instantiated as follows:

- The navigation context is set to $\langle n, R, \text{visited}(n) \rangle$ where $\text{visited}(n)$ is a navigation event related to destination unit $n$ and all rules triggered by the event $\text{BeforeLoad}(n)$ are executed.

- The root query of $n$ is evaluated (if it exists) using $R$ as a parameter.

- All compositions of $n$ are evaluated as follows:
  - The composition query is evaluated (typically) using the value $R$ as a parameter.
  - The child NU is instantiated using this procedure and a new temporary navigation context $\langle n_C, R_C, \text{visited}(n_C) \rangle$ where $n_C$ is the child NU, $R_C$ is the instance of the root concept of $n_C$ calculated by the composition query and $\text{visited}(n_C)$ is the corresponding navigation event. Before the instantiation all rules triggered by the event $\text{BeforeLoad}(n_C)$ are executed and after the instantiation all rules triggered by the event $\text{AfterLoad}(n_C)$ are executed.

- All rules triggered by the event $\text{AfterLoad}(n)$ are executed.

- Links are instantiated by evaluation of link query and by calculating the values of anchoring expressions. Anchoring expressions are transformed into the target format (e.g. HTML).

- Attributes are instantiated by evaluating the attribute expressions. Subsequently they are transformed into the target format.

4.5.3 Example AM

The purpose of this example is to demonstrate a partial specification of a (somewhat) meaningful Web application expressed using the Hera AM graphical notation, the corresponding RDFS graphs and the corresponding codes in the abstract syntax. Consider the sample domain described in Section 4.2. The AM presented here and depicted in Figure 4.18 allows PC members to assign papers to reviewers and allows reviewers to review papers. Presumably the domain database already contains instances of authors, reviewers, PC members and papers. In addition, we assume that every submitted paper has already an evaluation associated to it.

The initial NU is the $\text{Login}$ unit containing the $\text{LogForm}$ with a textual (uninitialized, without query) input $cid$. The event $\text{SetUser}$ triggers a rule initializing the $\text{session:user}$ resource that holds an instance of $\text{Person}$ (more precisely of its sub-class) who logged in. The RDFS graph of the $\text{Login}$ navigation unit with a navigation condition realized using $\text{AutoLink}$ is shown in Figure 4.19. The same is expressed using the abstract syntax as:
Figure 4.18: An example of AM using Hera AM notation

Figure 4.19: RDFS representation of the Login NU
NU Login() = {
    COMP Composition_ID1 = {THIS, LogForm}
}

FORM LogForm = {
    COMP Composition_ID2 = InputText UID;
    TRIGGER select = {
        AUTOLINK ALink_ID1 = {
            NU THIS,
            PATHS
            {
                PATH
                {
                    IF ( #R > 0 )
                    {
                        QUERY
                        {
                            SELECT { R }
                            FROM { {dm:Reviewer}<rdf:id>{R},
                                    {R}<dm:cid>{ID} }
                            WHERE { ID = uid }
                        }
                        NAVIGATE Reviewer.Main(R)
                    }
                    PATH
                    {
                        IF ( #PC > 0 )
                        {
                            QUERY
                            {
                                SELECT { PC }
                                FROM { {dm:PCMember}<rdf:id>{PC},
                                        {PC}<dm:cid>{ID} }
                                WHERE { ID = uid }
                            }
                            NAVIGATE PCMember.Main(PC)
                        }
                    }
                },
                EVENT SetUser(U),
                CAPTION "OK"
            }
        }
    }
}
The definition of the Login unit contains only a form specification. This form has only one input and one trigger with an associated autolink ALink_ID1 having two alternative navigation paths. Subclasses of the Composition concepts define nesting of AM elements (subclasses of Element — Composition_ID1 for nesting LogForm in the Login and Composition_ID2 nesting UID input in the form LogForm; note that for concise notation attribute and input composition do not have to be explicitly written in abstract syntax). AExpression_ID1 and AExpression_ID2 are conditional expressions identified in the code by the IF keyword. The trigger used in the form LogForm causes the event SetUser with the parameter uid (value of user ID) when it is activated by the user (the corresponding button is pressed). This event triggers the following rule that updates the session content.

```
RULE Init_User =
{
  ON SetUser(xsd:string U)
  IF {true}
  {
    INSERT
    {
      {session:session}<session:user>{U}
    }
  }
}
```

Elements of AM can be declared with a name (as it is for instance for EvForm, uid, or ALink_ID1), or without a name (as it is for instance for the trigger or the optional paths). The second approach can be used when the name is irrelevant and there is no need to refer to it. Note that the specification of the optional navigation paths done for demonstration of the model is presented completely here (the disjunction of conditions is not a tautology). The optional navigation paths defined by ALink_ID1 navigate to the Reviewer.Main NU or to the PCMember.Main NU. This solution of presenting proper information to the proper types of users is shown only for the sake of conditional navigation demonstration.

The Reviewer.Main unit contains a list of reviews assigned to a concrete reviewer (determined by the instance of the root concept) with forms allowing the reviewer to change the evaluation text by means of the Review.ForReviewer sub-unit. The Review.ForReviewer contains the Review.RevForm form for entering a new value of the review text. The instance of the root concept of this form is inherited from the Review.ForReviewer NU and the initial value of the input text is the original review text (or none). The form submission is associated with the ALink_ID autolink with only a single path. This path navigates to the Reviewer.Main unit that is a parent unit of the Review.ForReviewer. The RDFS representation of the Review.ForReviewer unit is shown in Figure 4.20. The same can be expressed using the abstract syntax as:
Figure 4.20: RDFS representation of the Review.ForReviewer NU

```
NU Review.ForReviewer(dm:Review AS R) =
{
    ROOT { R }
    ATTRIBUTE content =ut
    "The paper: " + {R}<dm:ofPaper>{}</dm:content>{C}
    }
    ATTRIBUTE title =ut
    "Title: " + {R}<dm:ofPaper>{}</dm:title>{T}
    }
    DCOMP ReviewForm =
    {COMP(TI, Review.RevForm(R)),
    }
    }
FORM Review.RevForm(dm:Review AS R) =
{
    ROOT { R }
    InputText ReviewText = { "Review: " +s {R}<dm:text>{T} }
    TRIGGER select =
```
The unit \emph{PCMember.Main} models a page for PC members with a set of papers that need to be assigned to reviewers (the \emph{Review.ForPC} unit is not described in detail here) and an assignment form where a PC member can assign a paper to a reviewer.

### 4.6 Conclusion

This chapter treats Research Topic 3 from Chapter 1. It explains the extension of the basic AM introduced in the previous chapter. The extended application modeling technique allows the specification of advanced Web applications with sophisticated application logic. The extension includes facilities for user input and the content manipulation model describing runtime updates of the data content. This model includes a flexible specification of the navigation structure (presentation logic) and an advanced specification of user feedback collection and processing together with data content manipulation. The content manipulation model and the navigation model are separated by means of events that are triggered in the Web interface (navigation model) and processed by rules in the content manipulation model. The support for Web Services allows a designed WIS to communicate with external systems in the Web environment.
Chapter 5

Modeling Collaboration in Web Information Systems

Chapter 4 describes the advanced modeling of WIS. AM determines the navigation structure and effects of actions taken by users on the system (data) state. This application modeling technique is navigation-driven where first the navigation view is defined for a domain and then possible data content manipulations triggered by the navigation interface are defined. However, if a designed WIS requires participation of multiple users involved in a complex business process the design of such a system driven primarily by navigation structure (and subsequent maintenance) may become difficult. The reason is that although the AM can capture navigation paths of involved different (user) roles implementing their tasks, it does not explicitly reflect the mutual synchronization of these tasks. If the (collaboration in the) business process changes it is hard to reflect the changes to AM because the process can only be indirectly derived from it.

This chapter introduces an improvement of the design method in the form of a collaboration model that can be used for the automatic generation of a proper AM (discussed in Chapter 4) for a WIS requiring collaboration between users. The collaboration model explicitly expresses synchronization and communication between different types of users performing different tasks. The contribution explained in this chapter addresses Research Topic 4 proposed in Chapter 1 and it is twofold: the extension of UML activity diagrams with a language (abstract syntax) for the detailed specification of particular activities to form a lightweight process model (called collaboration model) and the automated procedure for the generation of AM (presentation and business logic) from the process specification.

Section 5.1 revisits current research on business process modeling and presents an overview of the most used techniques. The motivation and position of the collaboration model used for WIS design is discussed in Section 5.2. The task model and collaboration model are explained in detail in Sections 5.3 and Section 5.4. The generation of AM from the collaboration model is described in Section 5.6. Section 5.7 concludes this chapter.
5.1 Overview of Process Modeling Techniques

Accurate specification of business processes is vital for the development of WIS (and IS) that control and run complex processes, especially when a number of participants is involved. Such process specification should explicitly model synchronization and communication of sub-processes in a transparent way. WIS are often running such business processes, as can be seen for example in the conference review and submission system discussed in this dissertation and that involve authors, reviewers and PC members. There are multiple process specification techniques ranging from formally well-founded approaches such as state machines and Petri nets [Reisig and Rozenberg, 1996] to methods where the accent is on their practical deployment like for example in Business Process Modeling Language (BPML) [Srivastava et al., 2002]. A brief summary of some selected methods is used in the following paragraphs in order to motivate the choice we have made.

State machines define systems in terms of their data states and transitions between the states. Transitions are labeled with events (and are fired when events occur) and define state changes. State machines are a well-understood approach, but fail to capture parallel and distributed processes. The most used state machine diagram technique is the UML state chart.

Petri Nets (PN) represent another well-formalized modeling technique that uses a graph-based specification of process structures and a token simulation semantics. PN allow to model parallel and distributed processes. The notion of states and transitions is changed compared to the semantics of state machines. Places replace states and represent containers allowing to store tokens. Transitions can be fired when all input places have tokens and transform them into output tokens. There are many variations of PN; one of the most used ones is colored PN, where tokens can have different colors (or can carry different data structures). PN are a well understood technique allowing simulation and verification of process models, but they are unfortunately rarely used in software development. Nevertheless, most of the techniques used in practice inherit some properties of Petri nets.

Process algebras (the best known types are Communicating Sequential Processes [Hoare, 1985] and Calculus of Communicating Systems [Milner, 1980]) represent another branch of well formalized approaches where process structures are represented by universal algebraic structures (concretely the group structures) and a set of algebraic operators can be applied to them. Process algebras allow reasoning about processes and their compositions by means of equational calculi, but they are rarely used outside the academic world.

BPML is an XML-based business modeling meta-language developed by the BPM initiative within the OMG group. It is a complex language supporting the use of Web Services and Web Service Choreography Interface (WSCI). It can use Business Process Modeling Notation (BPMN) as its graphical representation.

UML activity diagrams (AD) are a widely used modeling approach. It inherits some features of Petri nets (synchronization of parallel and distributed processes) and has a simple graphical notation. The formalization of AD semantics can be found in [Eshuis, 2002].
5.2 Motivation

Hera AM-NM described in Chapters 3 and 4 specifies the navigation structure by covering all possible navigation paths users can follow. The AM-CMM captures the effects of user interaction on the domain and context data and on the navigation structure itself. However, this specification is built from a user (role) perspective and it does not explicitly express the objectives of the designed system. Building a navigation structure perceivable by users is the key factor in the design. However, the user perspective is limited by the role he plays in the system. For instance, the author of a publication may not review and evaluate submissions (if he is not also assigned the roles of reviewer or PC member) and thus his role in the submission system is limited to sending submissions and receiving notifications. Since different involved roles have different tasks their views and interactions with the system are also different. The global view on the system business process reflecting the synchronization of particular tasks and participation of different user roles is missing in a solely navigation-driven design.

A simple collaboration model (CM) explicitly describing user tasks and their synchronization can, to a large extent, help to develop a suitable AM. In fact, if the tasks are defined at a sufficient level of detail and are based on activities a typical WIS performs (i.e. that can be expressed by AM), it is possible to automate the procedure of deriving the corresponding AM from the process model. Every information system should be able to interact with users and should be able to perform operations that may be hidden from users but have effects on this interaction. Analogically activities performed within a WIS (interacting with its users) can be divided into Interaction Activities where users participate and System Activities that are performed completely by the system and are expressed by a CM. This chapter presents an approach that allows the (semi-)automatic generation of NM from CM interaction activities and CMM from CM system activities. The most important step in NM generation is the partitioning of user tasks into clusters of activities that can be performed within a single browsing session and are not dependent on asynchronous communication with other users. The main step of CMM generation is transforming system activities into CMM rules and generating facilities (message queues) for the asynchronous communication that goes beyond the scope of single user sessions.

The discussed extension of the Hera method is the introduction of a CM together with automated procedure that transforms CM into a corresponding AM [Barna et al., 2006]. The Hera CM satisfies the following requirements:

- **Requirement 1:** It should have precise semantics that would avoid any ambiguous interpretation not only by designers but also by computers (it is supposed to be automatically transformed into AM).

- **Requirement 2:** It should be based on a known and commonly used method that would help to minimize the learning curve and that would avoid misinterpretations.

- **Requirement 3:** It should be easy to implement, especially considering the procedure of generation the Hera AM.
Most methods discussed in Section 5.1 (Petri Nets, Process Algebras, etc.) are well formalized and fulfill Requirement 1. Only few of them however are commonly used in software design nowadays Requirement 2:. We believe that UML — being the most commonly used software design technique — is also often used for (lightweight) specification of processes. UML activity diagrams have formalized semantics (see Section 5.1) and are easy to extend for our purpose Requirement 3. The extension involves a detailed specification of activities that can be easily matched with the Hera AM vocabulary.

The whole design method based on the CM specification is shown in Figure 5.1 and contains the following steps:

1. Identification of actors/roles (typically representing user groups or external systems) and their tasks within the collaboration with the system. This step results in a task model (TM) describing the mapping of actors to tasks.

2. Specification of collaboration processes hosted by the system in the form of a Collaboration Model (CM) where all actors and the system are involved in a collaborative process.

3. Specification of the application domain data structure that allows the detailed description of activities in CM resulting in a refined (detailed) CM.

4. Transformation of CM into AM. This transformation procedure is automated and it results in NM and CMM.

5. Specification of a presentation model defining layout of target Web pages (not part of this dissertation, details about presentation modeling can be found in [Fiala et al., 2004]).
5.3 Task Model

The functional specification of a system is determined by (business) goals that must be reached. These goals can be mapped to particular tasks that are performed by the system cooperating with users having different roles represented by actors. Actors represent all stakeholders that have some relationship with the designed system and share the same role in the system. The designed system itself is not considered to be an actor, but it has a special status. The common categories of actors are human actors, organizations, or external systems. The task model identifies actors and their tasks within the system and it can be expressed using a UML use case diagram. Figure 5.2 shows the use case diagram for the submission system. The specification of use cases is restricted to using only use cases (representing tasks), actors and their relationships. Special features as <<extends>>, <<uses>> and specialization properties are not considered in the use case specification, because a detailed description of use cases (tasks) is given in the CM. It is however possible to extend this method to fully exploit use cases, if appropriate mappings of the advanced features to the collaboration specification can be defined. The following definition formalizes the task model.

![Use case diagram for the submission system](image)

**Figure 5.2: Task model visualized using the use case diagram**

**Definition 5.1 (Task Model)** A task model of the system $S$ is a tuple $< A, T, \alpha, T_S >$, where:

- $A$ is a set of roles (actors) interacting with the system,
- $T$ is a set of actors’ tasks,
- $\alpha \subseteq A \times T$ is a relation determining the mapping of actors to tasks and
- $T_S$ is the set of system tasks.

Definition 5.1 specifies a task model as a set of actors, a set of tasks and their mapping. This structure can be easily expressed using use cases diagrams.
Example 5.1 (Task Model) The task model for the submission system from Figure 5.2 represented as $T_M(S) = \langle A, T, \tau, T_S \rangle$ would be:

\[
A = \{\text{Author, PCMember, Reviewer}\} \\
T = \{\text{MakeSubmission, AssignReview, EvaluatePaper, MakeReview}\} \\
\alpha = \{(\text{Author, MakeSubmission}), (\text{PCMember, AssignReview}), \text{PCMember, EvaluatePaper}, (\text{Reviewer, MakeReview})\} \\
T_S = \{\text{SubmissionFlow}\}
\]

5.4 Collaboration Model

The entire business process of the designed system is described by a collaboration model defining the system processes and the participation of actors in them. Not less important is the synchronization of activities captured by this model. A conference submission and review process for example is closely related to the life span of conference submissions, where all actors (authors, reviewers, PC members) perform their tasks in collaboration with the system (an author submits a paper, a PC member assigns reviewers and makes the final verdict and reviewers review the paper). The term collaborative system merges a process description with a distribution of activities to tasks. Please note that this definition is valid only in the context of the Hera method. Collaborative system is defined as a Labelled Transition System (LTS) — a state transition system expressing transitions between its states where the transitions can be associated with labels that are associated with events.

Definition 5.2 (Collaborative System) A collaborative system is a guarded LTS represented by the tuple $\langle S, E, G, \rightarrow, S_0, S_t, T_s, T, \tau \rangle$, where:

- $S$ is a set of all states,
- $E$ is a set of transition labels (events),
- $G$ is a set of boolean expressions (guards),
- $\rightarrow \subseteq S \times S \times E \times G$ is the transition relation,
- $S_0, S_t \subseteq S$ are sets of initial, respectively terminal states,
- $T_s$ is the (set of) system tasks(s),
- $T$ is a set of non-system tasks and
- $\tau \subseteq S \times (T_s \cup T)$ is the task relation mapping states to tasks.

Transitions are triggered (associated with) by events from $E$ if guarding statements from $G$ are satisfied,
The semantics of a Collaborative System (CS) from Definition 5.2 follows the semantics of UML activity diagrams (discussed in detail in Section 5.4.1). Unlike in state charts or Petri Nets, states here denote activities performed by the system or by actors. Activities last from the moment an associated activity state performing the activity is triggered until it is accomplished. Activity states can be triggered when preceding activity states are finished or they can be triggered by external events. CS can be visualized using UML activity diagrams (further AD). The submission CS in Figure 5.3 depicts all activities within the processing of a single submission. These activities are distributed to appropriate tasks belonging to actors and the system.

There are two types of transition labels (events) we consider in our CS representation: \( \overleftarrow{a} \) denotes a transition from an activity state \( a \in S \) and \( \overrightarrow{e} \) denotes waiting for an (possibly external) event \( e \). Guarding statements are logical expressions that decide on optional execution branches (decision blocks). The default value of the guard statement is true (and then it can be omitted). The notation for a transition would be \( a_1 \xrightarrow{\text{event[guard]}} a_2 \) meaning that the transition from the state \( a_1 \) to the state \( a_2 \) happens when \( \text{event} \) occurs and \( \text{guard} \) is satisfied. The shortcut notation for \( a_1 \xrightarrow{\text{true}} a_2 \) would be \( a_1 \xrightarrow{\} a_2 \).

The activity diagram in Figure 5.3 presents the conference submission CS. The sequence of activities is spread to the swimlanes representing the identified tasks of the actors. The activities performed by human actors (users of the WIS) are restricted to observing information offered by the system, entering information into the system and possibly communicating with different actors. This is a natural set of activities that a human user can perform using the (Web) computer interface. All other activities including data updates are under control of the system. This system-centric model, where all data manipulation activities are encapsulated within the system task, follows the typical (web) client-server scenario and minimizes the risk of unsynchronized data manipulations. The information that needs to be shared/exchanged between different tasks is passed as data objects (event parameters) associated with transitions. A transition from an activity state \( a \) to the following activity is denoted as the event \( \overleftarrow{a} \). The following example shows the structure of the CS from Figure 5.3 by means of LTS specified in 5.2.

**Example 5.2 (Collaborative System)** The excerpt of the CS specification from Figure 5.3 represented as the \(< S, E, G, \rightarrow, S_0, S_t, T_S, T, \tau >\) tuple would be:

\[
S = \{ \text{EnterAuthorLogin, StoreAuthor, EnterSubmission, ViewMissed,} \\
\text{ViewAckn, StoreSubmission, ...} \}
\]

\[
E = \{ \overleftarrow{\text{EnterAuthorLogin}}, \overleftarrow{\text{StoreAuthor}}, \\
\text{EnterSubmission, ViewAckn, ...} \}
\]

\[
G = \{ [\text{submission.date} > \text{Conference.subDeadline}], \\
[\text{submission.date} \leq \text{Conference.subDeadline}], \\
[\text{count(review.inEval.ofPaper)} > 1], [\text{count(review.inEval.ofPaper)} < 2] \}
\]
\[
\rightarrow = \{(\text{EnterAuthorLogin}, \text{StoreAuthor}, \text{EnterAuthorLogin}, \text{true}), \\
(\text{StoreAuthor}, \text{EnterSubmission}, \text{StoreAuthor}, \text{true}), \\
(\text{EnterSubmission}, \text{ViewAckn}, \text{EnterSubmission}, \\
[\text{submission.date} \leq \text{Conference.subDeadline}]), \\
\ldots \\
(\text{ViewAckn}, \text{ViewSubmission}, (\text{ViewAck}, \text{StoreSubmission}), \\
\text{true}), \ldots\} \]

\[S_0 = \{\text{start}\}\]

\[S_t = \{\text{end}\}\]

\[T_S = \{\text{SubmissionFlow}\}\]

\[T = \{\text{MakeSubmission, AssignReviewer, Evaluate, MakeReview}\}\]

\[\tau = \{(\text{EnterAuthorLogin, MakeSubmission}), (\text{StoreAuthor, SubmissionFlow}), \ldots\}\]

Note the representation of a decision using the \([\text{submission.date} \leq \text{Conference.subDeadline}]\) guard and the representation of a join by the composed event specifying the accomplishment of both parallel execution branches \((\text{ViewAckn, StoreCoAuthors})\).

### 5.4.1 Activity Diagrams Summary

In this section we take a closer look at the structure and semantics of the activity diagrams used in our method. Regarding the structure, in AD we consider initial and end states, activity states, synchronization primitives (parallel forks/joins and decisions/merges), wait states and edges describing state transitions. Transitions are labeled with events representing accomplishment of previous activity states, waiting for external events, or parallel activities. In addition, AD captures also the mapping of activity states to tasks of different actors. The considered basic AD elements used for CM are presented in Tables 5.1 and 5.2.

Besides the initial and end states all other states are activity states representing activities performed by actors or the system. They are triggered by accomplishing of preceding activity states and/or by an event with guard as it follows from a CS structure. Forks split parallel execution branches and they always must be joined later in the process. In order to join parallel activity branches correctly, the branches must be synchronized with a \textit{wait} state before the joining, where for last activities in parallel execution branches \(a_1, \ldots, a_n \in S\) waiting blocks are associated with the composed events \(\overline{a_2}, \ldots, \overline{a_n}\) for the branch ending with \(a_1\); then \(\overline{a_1}, \overline{a_3}, \ldots, \overline{a_n}\) for the branch ending with \(a_2\); etc. (For the details about composed events see Section 5.5.5). Another CM element is the decision that evaluates guarding conditions based on underlying data and variables and decides on one of the branches. Briefly summarized, the following properties must hold:
<table>
<thead>
<tr>
<th><strong>Author/MakeSubmission</strong></th>
<th><strong>System/SubmissionFlow</strong></th>
<th><strong>PCMember/AssignReviewer</strong></th>
<th><strong>PCMember/Evaluate</strong></th>
<th><strong>Reviewer/MakeReview</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter Author Login</td>
<td>Store Author</td>
<td>View Assignment</td>
<td>View Reviews</td>
<td>Enter Evaluation</td>
</tr>
<tr>
<td>Enter Submission</td>
<td>Author</td>
<td>Enter Assignments</td>
<td>Reviews</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Submission</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[submission.date &gt; Conference.subDeadline]</td>
<td>Store Submission</td>
<td>Store Review</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[count(reviews) &gt; 1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[count(reviews) &lt; 2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Store Evaluation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Evaluation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>View Submission</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>View Submission</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5.3: Example CS**
<table>
<thead>
<tr>
<th>Graphical Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="example" alt="Initial State" /></td>
<td>The <em>Initial State</em> is the first state of a CS (so there must be exactly one). The initial state has no incoming edges and one outgoing edge.</td>
</tr>
<tr>
<td><img src="example" alt="End State" /></td>
<td>The <em>End State</em> is the terminal state of a CS. A CS can have multiple terminal states. An end state has one incoming edge and no outgoing edges.</td>
</tr>
<tr>
<td><img src="example" alt="Activity" /></td>
<td>An <em>Activity State</em> is any state corresponding to a given activity. A system is in this state while the appropriate activity is being performed. An activity state can also be composite, which can be defined by a separate CS. An activity state has one incoming and one outgoing edge.</td>
</tr>
<tr>
<td><img src="example" alt="Wait State" /></td>
<td>A <em>Wait State</em> provides waiting for a specified event, and also provides correct synchronization of parallel execution branches when they are joined. A wait state has one incoming and one outgoing edge.</td>
</tr>
<tr>
<td><img src="example" alt="Decision" /></td>
<td>A <em>Decision</em> is a pseudo-state with virtual duration 0, when a condition (following from previous state) is evaluated and the appropriate branch is executed. A decision has one incoming and multiple (at least 2) outgoing edges.</td>
</tr>
<tr>
<td><img src="example" alt="Merge" /></td>
<td>A <em>Merge</em> joins alternative sequences of activities that have been split by decisions. A merge has multiple incoming edges and one outgoing edge.</td>
</tr>
<tr>
<td><img src="example" alt="Fork" /></td>
<td>A <em>Fork</em> splits sequences of activities that can be executed in parallel, or are independent. A fork has one incoming and multiple outgoing edges.</td>
</tr>
</tbody>
</table>

Table 5.1: Elements used in AD
5.4 Collaboration Model

<table>
<thead>
<tr>
<th>Graphical Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Join symbol]</td>
<td>A Join joins sequences of activities that are executed in parallel or independently and that have been split by a fork. Join has multiple incoming and one outgoing edges.</td>
</tr>
<tr>
<td>![Data object symbol]</td>
<td>Data object describing the parameter of event or message.</td>
</tr>
</tbody>
</table>

Table 5.2: Elements used in AD (continued)

- Optional execution branches that are split by a decision block must be at some point merged by a merge block.
- Parallel execution branches that are split by a fork block must be at some point joined by a join block.
- Parallel (independent) branches must end with a wait state before being joined. This ensures that the end of the execution of parallel branches is delayed until the slowest branch is finished.
- The disjunction of all conditions associated with a single decision block must be a tautology (always true) in order to prevent the execution to be hung at the decision block.

Because the semantics of AD have been previously formalized in few different ways (as we already mentioned) this paragraph contains only a brief informal summary. The term configuration refers to the overall state of a CS instance. This includes the data state and the state of all counters determining the actual states of the CS instance (it is possible that more activity or wait states are “active”). In this approach AD are understood as a special case of statecharts. Parallelism is modeled by using state tree hierarchies, where every state can have exactly one parent. The state nodes in the hierarchy can be of the type OR or of the type AND. In the first case the states represent a decision (choice) and one of the child state nodes must be in the configuration (all subtrees it represents) and in the second case they are parallel states and all of the child state nodes must be in the configuration. The Petri nets semantics is based on the mapping of AD to Petri nets such that state nodes are mapped to places and edges are mapped to transitions. Another variation is the semantics of Petri net token simulation (game). Another approach is based on the description of the AD semantics based on clocked transition systems. AD are decomposed to activity
hypergraphs (pseudo-states — decisions, merges, forks and joins are transformed to hyperedges) and execution of these hypergraphs is described using LTS. All approaches are in depth discussed in [Eshuis, 2002]

5.4.2 Expressing Collaborative Systems Using ECA Rules

The ECA rules mechanism originally adopted for active databases has appeared to be a convenient mechanism for describing behavior of reactive systems. ECA rules are used in the Hera advanced AM and they are also a natural way of expressing CS.

ECA rules can be represented by expressions $e[c] \rightarrow a$, where $e$ is an event triggering the activity $a$ if the condition (guard) $c$ is satisfied. As mentioned previously, actions are modeled in AD (and thus also in CM) as activity states, conditions as decision blocks and events are represented as transitions between activity states. Moreover, external events (such as waiting for required information) are represented by $\text{wait}$ blocks.

In order to correctly describe the structure of CS using ECA rules (in order to express guarded $\neg a$ events meaning accomplishing of activities) we need to introduce composite events of the following types:

- A **joined event**, constructed using the join operator ($\cdot$), is of the form $e_1, e_2, ..., e_n$ and takes place after all events $e_1, ..., e_n$ take place regardless of their order.

- A **sequenced event**, constructed using the sequence operator ($;$), is of the form $e_1;e_2;...;e_n$ and takes place when all events $e_1, ..., e_n$ take place in the order they are written.

- A **merged event**, constructed using the merge operator ($|$), is of the form $e_1|e_2|...|e_n$ and takes place when at least one of events $e_1, ..., e_n$ has taken place.

Guarding conditions should be coupled with single (not composite) events and in the case of composite events they are, for every single event in a composite event, evaluated immediately at the time the single event occurs. This is a straightforward requirement, because the system data state, (possibly) queried by guarding condition expressions, may differ in time.

A. Splitting Activity Sequences

Splitting of activity sequences (referred to as fork) is expressed by sets of ECA rules triggered by a single event. The example in Figure 5.4 demonstrates different possible situations that can occur when triggering parallel execution branches and their expressing using ECA rules. Event $e_1$ triggers parallel execution branches that can start any activity state, wait state, another parallel execution (branch), or decision. In the case of wait state a possible subsequent activity state is executed, when the sequence of the fork triggering event and the event associated with the wait block $e_1; e_2$ have taken place. If another parallel execution is triggered in one of the parallel branches ($A_3$ and $A_4$), this is considered as another parallel branch belonging to the original fork. If a decision is following, alternative
5.4 Collaboration Model

execution (with activity states $A_5$ and $A_6$ in the example) is given by the guards $[c_1]$ and $[c_2]$.

![Diagram of activity states and guards](image)

Figure 5.4: Triggering parallel execution branches

B. Joining Activity Sequences

Joining of activity sequences (referred to as join) can be expressed by a single rule as shown in Figure 5.5. This rule is then triggered by a composite event $e_1, e_2, ..., e_n$ that expresses the fact that all events $e_1, ..., e_n$ took place and this composite event takes place at the same time as when the last of the particular events takes place.

![Diagram of composite event and activity](image)

Figure 5.5: Joining parallel execution branches

C. Branching Activity Sequences

Branching of activity sequences (referred to as decision) triggers alternative execution branches by evaluating single guarding conditions. As already mentioned, every branch has its own guarding condition and to ensure the correct execution the conditions must be exclusive and their disjunction must always be satisfied. Figure 5.6 shows a decision block with different possible execution branches and its representation using ECA rules. The interesting situation arises when a decision block after the $e_1$ event is followed by a
wait state associated with $e_2$ event. In case we neglect eventual data updates caused by other CS instances the sequence of the events $e_1; e_2$ and the evaluation of the condition $[c_2]$ would be arbitrary, so the optional execution branch would be expressed by the rule $e_1; e_2[c_2] \rightarrow A_2$. Unfortunately, one can not rely on the fact that (shared) data is not updated until the event $e_2$ and so the possible evaluation of condition $[c_2]$ can be different before $e_2$ and after it. The rule $e_1[c_2]; e_2 \rightarrow A_2$ expresses the fact that the rule is triggered by the composed event $e_1; e_2$, but the condition $[c_2]$ must be evaluated before $e_2$. In case a decision is followed by another decision, the guarding condition of subsequent alternative execution branches is the conjunction of their guarding conditions. This is also one of the reasons why the appropriate guards in composite events must be evaluated when the associated single event occurs.

![Diagram](image)

Figure 5.6: Triggering optional execution branches

### D. Merging Activity Sequences

Merge blocks represent proper merging of alternative execution branches. They can be expressed as a single ECA rule with a composite event representing that at least one of named events took place. Figure 5.7 shows the example of the merge block where the composite event $e_1[e_2]...e_n$.

![Diagram](image)

$$e_1[c_1] \rightarrow A_1$$
$$e_1[c_2]; e_2[true] \rightarrow A_2$$
$$e_1[c_3] \rightarrow A_3$$
$$e_1[c_4] \rightarrow A_4$$
$$e_1[c_4 \land c_5] \rightarrow A_5$$
$$e_1[c_4 \land c_6] \rightarrow A_6$$

Figure 5.7: Merging alternative execution branches
5.5 Specification of Activities

A CS expressed by CM that specifies only activities as black boxes is not sufficient for the generation of a target AM. The specification of activities together with information they produce or consume is needed for a specification, needed for AM generation. An activity state can represent a whole CS (composed activity state), or can represent a single (atomic) activity. The example in Figure 5.3 is designed in such a way that all activity states are atomic. However, any activity can represent a nested CS and in this case the nested CS must be specified. In this recursive manner the complete CS model is specified in such a way that all activity states are completely defined — possibly by their decomposition into atomic activity states, that are of the following types:

- **Information observation**, wherein particular information is served to actors by the system. The observation activity states should be included in the tasks of actors.

- **Information entry**, wherein information required by the system is entered by actors. The information entry activity states should be included in the tasks of actors.

- **Information updates**, wherein the information content defining the domain and the system state are updated. The information update activity states should be included in the system task.

5.5.1 Information Observation

Within the information observation activities actors (users) perceive information offered by the system. In the process of AM generation the observation activities are transformed to the specification of a (set of) web page(s) offered to the users, or another way of communication such as sending emails or sending SOAP messages. However, the CM abstracts from the specification of the form in which the actors receive the information, which is the subject of the application modeling and it is decided within the transformation process. The specification of the information observation activities contains the description of the presented content.

**Example 5.3 (Information Observation)** The specification of the information observation activity ViewAckn displays the acknowledgement message and uses the parameter aname (the name of the author). It has the form:

```
ACTIVITY: ViewAckn(String aname)=
{
    OBSERVE
    {
        CONST("Dear " + aname + ", your submission has been registered.")
    }
}
```
The ViewAckn information observation activity displays the acknowledgement message together with a parameter (the name of the author). Note that there is no ViewAckn event specified. The expiration of the activity (transition to the subsequent activity state) depends on the user and there is no need to specify the default event ViewAckn. In the AM implementation of workflows however the events are specified in the form of links, forms, etc.

5.5.2 Information Entry

Within the information entry activity actors (users) provide information required by the system task. The specification of these activities contains the definition of the input data structure. The information can be gathered from non-human actors (e.g. external systems) by using Web service calls, or from human actors by entry forms.

Defining entry forms for information entry is almost equivalent to the form specification in the Hera AM (see Requirement 3: in Section 5.2). The only difference is that Hera AM forms do not specify which event is triggered (this is given by the workflow structure). An entry form consists of a number of inputs with defined cardinality (set denoted by "*" or a single value), data types (xsd types, concepts from CM and the RECORD type — locally defined data structure), input types — selection from predefined values (INPUTSEL1) and text field input (INPUTTEXT). A predefined value (or set of values) can be retrieved from the domain database using a query, or can be assigned using parameter values or constants. The information entry activities may have input parameters that serve as predefined values of inputs as has been mentioned and the default output parameter that is the completed set of inputs. The following example demonstrates the specification of the EnterSubmission activity using the abstract syntax.

Example 5.4 (Information Entry) The EnterSubmission activity specification demonstrates abstract syntax constructs used for the descriptions of inputs. The submission entry form is the parameter of the EnterSubmission event. It consists of the following inputs: ptitle, track, author, coauthors, file and date.

ACTIVITY: EnterSubmission(form:logininfo 1)=
{
    ENTER FORM submission=
    {
        INPUTTEXT ptitle = { LABEL("Enter title:") },
        INPUTSEL1 track =
        {
            QUERY
            {
                SELECT{ T }
                FROM{ {dm:Track AS T} },
                DISPLAY{ {T}<dm:tname> },
            }
        }
    }
}
5.5 Specification of Activities

The TRANSFERS keyword denotes the specification of the output parameter that is transferred to a subsequent activity state. Input forms defined within the information entry activities are (often) processed within the subsequent activities. Accessing information in the completed forms is simple and it is demonstrated in the following section.

5.5.3 Information Updates

Information update activities manipulate the domain data, or any kind of context data (for instance they can transform input parameters into different output parameters). Information updates are again analogical to updates used in the content manipulation sub-model of the Hera AM. For more details see Chapter 4.

Example 5.5 (Insert Operation) An example of an insert operation is the StoreSubmission activity, where a new instance(s) of the Paper and Author concepts are created. Note that this complex insertion is decomposed in the engine into a set of simple insertion operations.

ACTIVITY: StoreSubmission(form:submission s)=
{
    INSERT
    {
        {#new(dm:Paper) AS P}
        <#new(dm:written_by)>{#new(dm:Author) AS A};
        <#new(dm:written_by)>{#new(dm:Author) AS CO},
        {CO}<#new(dm:wrote)>{P},
        {A}<#new(dm:wrote)>{P},
        {P}<#new(dm:for)>{T},
        {T}<#new(dm:all)>{P},
        {P}<#new(dm:title)>{title},
        {CO}<#new(dm:firstName)>{cofname},
        <#new(dm:lastName)>{colname},
        <#new(dm:affiliation)>{coaffil},
        {A}<#new(dm:firstName)>{afname},
        <#new(dm:lastName)>{alname},
        <#new(dm:affiliation)>{aaffil}
    }
    WHERE
    {
        title = s.ptitle AND
        file = s.file AND
    }
}
FOR{ coauthor IN s.coauthors AS }
{
    cofname = coauthor.aname,
    colname = coauthor.asurname,
    coaffil = coauthor.affil
} AND
afname = s.author.aname AND
alname = s.author.asurname AND
aaffil = s.author.affil AND
T = s.track
}

An insert operation specifies the new data graph (added resources and attributes) in the \textbf{INSERT} clause and the specification of bindings for new values of attributes in the \textbf{WHERE} clause. Note the \textbf{FOR} keyword determining the iteration for all members of the set \texttt{s.coauthors}. This is an alternative to a retrieval query and the \textbf{ALL} operator. This binding of a set of values enforces the creation of multiple \textit{Author} concept instances (coauthors of the submission since the author is already stored).

5.5.4 Defining Events

Types of events that are defined in CM are overlapping with types of events used in AM and discussed in Chapter 4. There are no \textit{navigation events} because CM does not explicitly describe a navigation structure (it is only derived from it in a later step). Events not contained in AM are \textit{activity events} that are in CM associated with activity transitions. In summary the following types of events are distinguished in CM:

- \textit{Activity events} associated with activity transitions. Events of this type are identified by names of activities triggering it. A special case of activity events is \textit{message} that is associated with transition between activities from different tasks. They may serve the purpose of communication between different tasks.

- \textit{Temporal events} are identical to AM temporal events discussed in Chapter 4 and

- \textit{Exceptions} are identical to AM exceptions discussed in Chapter 4.

\textit{External events} as they are defined for AM are considered in the context of CM only when the external system triggering them is modeled as an actor in TM. In this case however these events become messages.

A. Activity Events

Activity events are transitions from one activity state to another. They can consume information defined as parameters and send information defined as an output value (graph) in the \texttt{TRANSFERS} part of the activity specification. This information transfer is unlike
transfer via messages performed in a synchronous manner which means the information would be exchanged at the moment one activity is accomplished and its subsequent activity is triggered.

B. Messages

Messages are activity events originating in one task and triggering an activity state(s) in another task(s). They serve as communication channels between the system and the external (human or non-human) actors. Because the external actors are by default considered to be asynchronous (it is safer to assume that the processing of messages is not immediate and can take an unpredictable amount of time, for example a reviewer needs some time to deliver a required review), a special mechanism is needed for handling the message sending and reception. For this purpose message queues are used and service rules are added to models describing the application logic. Another reason to have message queues is that a reactive (web) system typically executes multiple instances of a CS at the same time, which leads to multiple messages being sent to (potentially) the same actor instance (a user or an external system). The detailed description of message queues and their handling is given in Section 5.6.1.

C. Event Parameters

Because events can carry parameters, it is important to discuss how parameters of composite events can be retrieved. Note the difference between the composite events that take place if all participating (partial) events take place (joining of execution branches) and the composite events that take place when only some of their participating events take place (merging of execution branches). In the first case all parameters of partial events are available, but in the second case not. This leads to parameter constraints for the following composite events:

- In joined events all partial events must take place and so all parameters are available and there is no constraint.
- In sequenced events all partial events must take place as well and so all parameters are available and there is no constraint.
- In merged events an unknown participating event causes the event firing, so all participating events should share the parameter structure (or part of it) and the event providing the actual parameter is determined in runtime.

5.5.5 Expressing ECA Rules

ECA rules defined for CM are similar to AM rules. The only difference is the extension for the specification of activity events. Rules contain the specification, or a reference to their triggering event, condition and action. Concretely, the abstract syntax specification of ECA rules contains:
the rule ID,
• the triggering event ID or specification in the case of a composite event,
• the rule guarding condition that may contain queries retrieving acquired information from the data content and
• a reference (or specification) of the activity state (rule action) together with the specification of its input parameter(s).

Consider activity to be the action of a rule R. Then this rule also automatically generates the event ←−−−−− activity) when the activity is accomplished. This is the default event generated by every rule when the action is accomplished and it provides a simple mechanism for correct rule triggering. A completed activity is denoted as finished(activity). Event parameters are then referred as event.parameter. The excerpt of the grammar for defining rules using the abstract syntax is:

\[
\text{\langle eca-rule-def \rangle ::= 'RULE' \langle identifier \rangle '=' '{' \langle event-condition \rangle \langle rule-action \rangle '}'}
\]

\[
\text{\langle event-condition \rangle ::= \langle event-condition1 \rangle ( \langle event-operator \rangle \langle event-condition \rangle )* ('PARAMETERS' '{' \langle par-matching \rangle '}' )?)}
\]

\[
\text{\langle event-condition1 \rangle ::= 'ON' '{' \langle event-identifier \rangle '}' 'IF' '{' \langle condition \rangle '}'}
\]

\[
\text{\langle event-identifier \rangle ::= \langle identifier \rangle | 'finished' '{' \langle activity-identifier \rangle '}'}
\]

\[
\text{\langle event-operator \rangle ::= ',' | ';' | '}'}
\]

\[
\text{\langle condition \rangle ::= \langle bool-expression \rangle}
\]

\[
\text{\langle bool-expression \rangle ::= \langle bool-expression1 \rangle ( \langle retrieve \rangle )*}
\]

\[
\text{\langle bool-expression1 \rangle ::= \langle bool-const \rangle | \langle bool-var \rangle | \langle un-operator \rangle \langle bool-expression1 \rangle}
\]

\[
| \langle bool-expression1 \rangle \langle rel-operator \rangle \langle bool-expression1 \rangle}
\]

\[
\text{\langle rule-action \rangle ::= 'ACTION' '{' \langle action-identifier \rangle '}' | \langle action-def \rangle}
\]

\[
\text{\langle action-def \rangle ::= \langle observe-action \rangle | \langle entry-action \rangle | \langle update-action \rangle}
\]

The conditional expressions can refer to the domain data and to parameters of the triggering event. Moreover, they can contain arithmetic expressions, constants and binding variables that are initialized and can be used multiple times within a single rule.
5.6 Generation of Application Model

Example 5.6 (ECA Rule) Consider the CS for PC member in Figure 5.3. The ECA rule for the ViewSubmission activity would be:

```
RULE Rule_ViewSubmission = {
    ON { finished(StoreSubmission) AS s} IF { true } |
    ON { finished(StoreReview) AS r} IF { c > 1 }
    {
        QUERY
        {
            SELECT { count(REVS) AS c }
            FROM
            {
                {REV}<cm:inEval>{cm:Evaluation}<cm:hasReviews>
                {cm:Review AS REVS}
            }
            WHERE{ REV = r.review }
        }
        PARAMETERS { p = s.submission | r.submission }
        ACTION { ViewSubmission( p ) }
    }
}
```

Conditional expressions are boolean expressions. Constants, items of event parameters and values retrieved from the domain database can be used there.

5.6 Generation of Application Model

The AM generation process is an automatic transformation of the CS specification (CM) into a specification (or executables) that can be directly deployed on the host engine, in the discussed case the Hera AM (Figure 5.8). The CMM sub-model is generated from the system tasks that contain all update activities. Additional business rules providing user identification and message exchange are also added (details are explained in Section 5.6.1).
NM sub-model is generated from the specification of actor tasks. A primary navigation path for every actor is based on the structure of his tasks (details are explained in Section 5.6.2). The relationships of CM and AM are shown in Figure 5.8. The generation of this AM consists of the following steps:

- **Generating the content manipulation model** in the form of business (ECA) rules in the target AM based on the system tasks. This model includes context data maintenance rules for providing facilities for asynchronous communication between different actors using a Web interface.

- **Generating the navigation model** in the target AM based on the structure of CM activities. Optional heuristics for this can be selected.

### 5.6.1 Generation of the Content Manipulation Model

ECA rules that form CMM are generated from the detailed specification of activities that are also expressed by ECA rules. The intentional similarity between CM and CMM reduces the transformation of rules to transformation (resolution) of composed CM events into events triggered in NM. Transformation of rules is however only one of two major steps in the CMM generation. Context data (message queues, user identification etc.) allowing asynchronous communication between users are maintained by content manipulation rules that are also generated in this process.

In order to determine which information should be stored in context data and how to maintain this information we need to realize that Web applications of the type we discuss in this chapter must typically handle multiple instances of the same collaborative process represented by CM at the same time. For example, in the submission system a PC member within the activity state *ViewSubmission* would rather see all new submissions. The submission collaborative process is associated with a single submission, but the real system works with all submissions. The related problem is the aforementioned asynchronicity of the system task and actor tasks. For example, when the submission system sends the *Submission* message to a concrete PC member, he does not have to be working with the system at the time, etc. These issues must be treated by the AM and additional context model. In summary, the following problems must be solved using additional context data and content manipulation rules:

- **Queueing of messages.** Due to the asynchronicity in exchanging messages between the system and its users there is a need to generate message queues and appropriate rules allowing correct asynchronous message exchange between the system task and tasks of actors implemented in the AM navigation sub-model.

- **Processing of composite events.** Composite events are not directly caused by the system environment, but their occurrence and parameters must be calculated at runtime based on their component events. For this purpose there is a need to generate appropriate data structures and processing rules.
5.6 Generation of Application Model

- Identification of users. In case the user identification (their classification to roles) and registration is not expressed in CM there is a need to generate appropriate rules allowing it. In the submission example there are activities for identifying and storing authors, but not for identifying or storing other users.

All auxiliary data structures are specified in a so called context data model (CDM) that is maintained by automatically generated rules. The simplified CDM structure is shown in Figure 5.9.

A. Message Queues

The majority of user tasks (or their regions) represent the transformation of incoming messages to outgoing ones, in which some additional information is added (added value). For example, a PC member receives a submission’s information, adds assignments of reviewers to it and sends it back to the system. Note that these messages are asynchronous, because recipients (typically human actors) might not be already logged in on the submission system at the time the messages are sent. Furthermore, since there are possibly multiple instances of such process running at the same time (e.g. multiple submissions are being processed), also multiple instances of the same type of messages can be sent. This typical asynchronous communication problem is resolved by using message queues that store sent messages until they are accepted by recipients. The message queues and message queue maintenance rules are automatically created in the process of AM generation. The message queues represented as $MQueue$ concept in CDM are instantiated as singletons for all communication channels between the system and actor tasks. Figure 5.10 shows the $MQSubmission$ queue that stores $Submission$ messages sent from the system task to the $AssignReview$ task of PC members and the $MQAssignments$ queue storing $Assignments$ messages sent back.

Because of the asynchronous nature of the communication with actors, a concrete process is delayed when a recipient of a message is not ready to consume the message (and the message is put into a queue). Using queues allows to run multiple instances of the same process at the same time. The state of message queues determines the state of the instances in any given moment. When a user logs in on the system, messages that have
Figure 5.10: Message queue example

been sent to activities of his task but that have not been received (consumed) are made available to them so they can be processed. After a user logs out, all unprocessed messages are returned into appropriate message queues. For example, when a PC member logs into the system, all new Submission messages from the MQSubmission queue dedicated to the concrete PC member are presented to him. He can assign reviewers to a number of submissions (possibly all) and leave the rest. Then all assigned submissions are sent to the MQAssignments queue and are immediately processed by the system (appropriate process instances continue their execution). When the PC member logs off, all unassigned submissions are returned to the MQSubmission queue. The following operators will help to explain the rules providing the queue maintenance:

- \( \text{get}(\text{queue} : Q, \text{user} : U) \) returns a first appropriate (matching a concrete user \( U \)) message from a queue and at the same time removes it from the queue,

- \( \text{put}(\text{queue} : Q, \text{msg} : I_n) \) inserts a message \( I_n \) to the queue and

- \( \text{empty}(\text{queue} : Q, \text{user} : U) \) checks if the queue \( Q \) contains any messages for the user \( U \).

The operations are transformed into CDM data retrieval queries (described in Section 5.6.2) into data update business rules. The following situations (events) are handled:

- The system sends a message \( M \) to a user (his AM instance). A query is added in AM to every queue providing the operation \( \text{put}(Q, M) \). The identification of the user is included in one of the rules.

RULE Rule_ViewSubmission = {
    ON \{ finished(StoreSubmission) AS s} IF \{ true \} |
    ON \{ finished(StoreReview) AS r}\}
    IF \{ c > 1 \}
    { 
        QUERY
The Rule ViewSubmission rule is transformed into two rules (because of the composed event, for detail see next section) in CMM that implement put\((Q, M)\). The first one is (the second one analogical with different event):

RULE Put_MQSubmission =
{
  ON { finished(StoreSubmission) AS s}
  IF { true }
  {
    INSERT
    {
      {#new(cdm:Message)}<#new(cdm:queue)>{Q};
      <#new(cdm:params)>{s};
      <#new(cdm:for)>{U};
    }
    WHERE { Q=cdm:MQSubmission AND U=#user }  
  }
}

- A user logs in. For this user the state of message queues are checked and the navigation structure is updated accordingly (see Section 5.6.2). The following generated rule implements get\((Q, U)\) and it is triggered when appropriate message parameters are presented to the user, in this case Submission instances to a PC member:

RULE Get_MQSubmission =
{
  ON { afterLoad(NU_Submission) AS s}
  IF { true }
  {
    REMOVE
    {

This rule removes all messages with parameters that are presented in the navigation unit NU_Submission.

• A user sends messages to the system. In this case two sets of rules are added to CMM. The first set is generated for every incoming (from the point of view of users) message queue that puts unprocessed incoming messages back to the input queue (implements \( put(Q_{In}, M_{In}) \)).

RULE Return_MQSubmission = {
    ON { Assignments(cdm:Assignment A) }
    IF { true }
    {
        INSERT
        {
            {#new(cdm:Message)}<#new(cdm:queue)}{#new(cdm:MQSubmission)};
            <#new(cdm:params}){#new(cdm:Parameter) AS P};
            <#new(cdm:for}>U
        }
        FROM
        {
            {P}<cdm:has}>A,
            {A}<cdm:reviewer}>R
        }
        WHERE { U = #User AND R NOT IN SELECT * FROM dm:Reviewer }
    }
}

This rule represents returning of unprocessed messages back to input queues, in this case returning empty review assignments (without reviewers yet specified) to the MQSubmission queue. This rule is triggered on the submission of the assigning form (see NM in Section 5.6.2).

The second set of rules pushes processed messages that should be sent to an activity of a different task into appropriate message queue (sending of messages). An example if such rule implementing \( put(Q_{Out}, M_{Out}) \), in this case \( put(MQAssignments, A) \) for submission with assigned reviewers is:
5.6 Generation of Application Model

As follows from the previous explanation the state of every process instance (implemented as an AM instance in Hera) is given by the content of queues in which the system puts its messages. If all queues in which the system puts messages are empty, the system is not waiting for any message from users and then there is no running process instance (or it is only under system control). The exceptions are user messages that initiate a new process instance, in the case of the submission system it is the LoginInfo message from the Author role.

B. Resolving Composite Events

Composite events that are used to express synchronization primitives in CM are not triggered by a single environment event (e.g. user action) but are triggered by multiple single events. Hence, composite events are defined by means of single event expressions. They must be resolved (triggered with proper parameters) at runtime, which is done by monitoring single events and based on composite event expressions eventually composite events are triggered. The composite events are handled by the following mechanisms:

- *Joined events* are stored in a container. The composite event is generated when all participating events are in the container and the required event parameter is created as the union of all parameters of events in the container. For every joined event in CM a singleton of EContainer in CDM is generated. For the joined event \( e_1, e_2, \ldots, e_n \) \( n+1 \) rules are generated, where first \( n \) rules add the event to the container and trigger a rule that checks the number of rules and if the container is full the composite event...
is triggered. It can be expressed as:

\[ e_1[g_1] \rightarrow \text{store}(e_1), \text{trigger}(\text{check}) \]
\[ e_2[g_2] \rightarrow \text{store}(e_2), \text{trigger}(\text{check}) \]
\[ \ldots \]
\[ e_n[g_n] \rightarrow \text{store}(e_n), \text{trigger}(\text{check}) \]
\[ \text{check}[\text{nostored} = n] \rightarrow \text{trigger}(\text{composed}) \]

For the event \[ \text{StoreSubmission}, \text{StoreReview}[\text{count}(\text{reviews}) < 2] \] the following rules are generated:

**RULE Ev Resolver 0101**

\[
\{ \\
\text{ON} \{ \text{finished}(\text{StoreSubmission}) \text{ as } S\} \\
\text{IF} \{ \text{true} \} \\
\{ \text{INSERT} \\
\{ \\
\{\text{#new(cdm:Event)} AS E}\text{<#new(cdm:containers)}\{C\}, \\
\{\text{#new(cdm:params)}\{S\}; \\
\{\text{#new(cdm:for)}\{#user\}, \\
\{C}\text{<#new(cdm:evs)}\{E\} \\
\} \\
\text{WHERE} \{ C = \text{cdm:ev-resolver_01 } \}, \\
\text{TRIGGER} \{ \text{EvCheck01} \} \\
\}
\}
\]

**RULE Ev Resolver 0102**

\[
\{ \\
\text{ON} \{ \text{finished}(\text{StoreReview}) \text{ as } R\} \\
\text{IF} \{ c > 1 \} \\
\{ \text{QUERY} \\
\{ \\
\text{SELECT} \{ \text{count(REVS)} AS c \} \\
\text{FROM} \\
\{ \\
\{\text{REV}\text{<cm:inEval}\{cm:Evaluation\}} \\
\text{<cm:hasReviews}\{\text{cm:Review AS REVS}\} \\
\} \\
\text{WHERE} \{ \text{REV} = R.\text{review } \}
\} \\
\text{INSERT}
\}
\]
{#new(cdm:Event) AS E}<#new(cdm:containers)>{C},
<#new(cdm:params)>{R};
<#new(cdm:for)>{#user},
{C}<cdm:evs>{E}
WHERE { C = cdm:ev-resolver_01 };
TRIGGER { EvCheck01 }
}

RULE Ev_resolver_check_01 =
{
   ON { EvCheck01 }
   IF { c = 2 }
   {
      QUERY
      {
         SELECT{ count(E) AS c }
         FROM { {cdm:ev-resolver_01}<cdm:evs>{E} }
      }
      TRIGGER { Composite_01 }
   }
}

• **Sequenced events** are processed in the same way as joined events, but the order is also taken into account. For the sequence of events $e_1; e_2; \ldots; e_n$ rules are generated that store single events only when the event with lower order number has occurred (events are in the correct order). This can be expressed as:

\[
\begin{align*}
e_1[g_1] & \rightarrow \text{store}(e_1) \\
e_2[g_2 \land \text{stored}(e_1)] & \rightarrow \text{store}(e_2) \\
\vdots \\
e_n[g_n \land \text{stored}(e_{n-1})] & \rightarrow \text{store}(e_n), \text{trigger}(\text{composed})
\end{align*}
\]

• **Merged events** are decomposed to a set of rules, where every rule is triggered on a partial event. For the merged event $e_1; e_2; \ldots; e_n$ this can be expressed as:

\[
\begin{align*}
e_1[g_1] & \rightarrow \text{trigger}(\text{composed}) \\
e_2[g_2] & \rightarrow \text{trigger}(\text{composed}) \\
\vdots \\
e_n[g_n] & \rightarrow \text{trigger}(\text{composed})
\end{align*}
\]
C. User Identification

Within WIS specified by CM it is necessary to register new users, recognize returning users and determine their roles in the system. The user identification (and possibly first their registration) takes place at the beginning of every session. The general procedure is to check if the user is already registered and, if not, to register him and his role(s). For this purpose the two following rules are generated:

RULE Loggin_Find =
{
  ON
  {
    Login(xsd:string fname,
    xsd:string lname,
    xsd:string aff,
    rdfs:class role)
  }
  IF { c > 0 }
  {
    QUERY
    {
      SELECT{ count(U) AS c } 
      FROM
      {
        {U}<dm:firstName>{fname};
        <dm:lastName>{lname};
        <dm:affiliation>{aff};
        <rdf:type>{role}
      }
    }
    ACTION { #setUser(U) }
  }
}

for find(U) and

RULE Loggin_Register =
{
  ON
  {
    Login(xsd:string fname,
    xsd:string lname,
    xsd:string aff,
    rdfs:class role)
  }
  IF { c = 0 }
  {
    
  }

5.6 Generation of Application Model

5.6.2 Generation of the Navigation Model

Tasks of actors are in general transformed into the navigation structure. Tasks are divided into regions called activity clusters (AC) that can be performed in a single session and of which the states are determined by the input message queues. As an example we take the task of an author shown in Figure 5.11. Thick arrows in the figure represent events from outside (from the system) that in fact trigger activity state transitions. They are divided into non-blocking (depending only on fast activities of the system) and blocking (also depending on other actors). The blocking activity in the figure represents activities from tasks *AssignReviewer*, *Evaluate* and *MakeReview* of different actors. The blocking activities determine the partitioning of activity sequences into clusters.

Activity clusters are used to build a first iteration of the navigation structure. In this iteration activity clusters are transformed into navigation clusters (groups of NM elements). Navigation clusters represent portions of navigation space that can be browsed by users without limitations caused by blocking, possibly caused by waiting for messages from other users. Automated transformation of the CM to NM is based on a heuristic method that has been proposed considering the following:

- There are multiple types of users (actors) of the system and there is a need to identify a concrete user and his role in the system prior to his collaboration with the system. For instance, it is important to recognize roles of users of the submission system.

- The different types of users are typically involved in multiple instances of the process at the same time. The system then stores the state of all unfinished tasks of all users.
For example, in the submission system there are typically multiple submissions that must be reviewed and evaluated.

- The users can only perform (instances of) tasks that are not blocked by other users (roles). For example, a PC member can not make the final evaluation of a submission until all reviews are done.

Figure 5.12 shows the skeleton of a NM that is generated. The entry point is a login unit that requests data to identify users. The Login event triggers user identification/registering rules described in Section 5.6.1. The Actor.menu unit is specialized for every actor defined in TM and contains a set of menu items — links to all navigation clusters (derived from activity clusters) of tasks of every actor. For example, the unit Author.Menu would have two links, the first navigates to the submission cluster (Cluster 1 in Figure 5.11) and the second navigates to evaluation(s) (Cluster 2 in Figure 5.11). Moreover, as explained before, the current state of process instances (and tasks) is given by the state of input message queues. Hence, the access to navigation clusters is determined by the conditions evaluating the content of the message queues corresponding to activity/navigation clusters. For example, the condition denoted as NOT empty(MQEvaluation, user) determines the access of author user to his evaluation cluster from MakeSubmission task, if the queue MQEvaluation contains some messages (evaluations). In a real model this condition would have the form:

```java
... IF { c > 0 } {
  QUERY {
    SELECT{ count(M) AS c }
    FROM { {cdm:MQEvaluation}<cdm:msgs>{M} }
  }
}
...```
5.6 Generation of Application Model

A. Partitioning of Tasks

It is important to realize that nothing can be said about synchronicity (in this case promptness) of actors responses, because this is determined by the manner (e.g. time schedule) in which the actors use the system. A response in this case corresponds to the time in which a particular business entity is delivered to the system from an actor. For example, this is the time between \textit{StoreReviews} and \textit{EnterEvaluation} in the submission CS, in which the human actor of the type \textit{PCMember} delivers \textit{Evaluation}. For the sake of clarity we introduce another notation for activity states that determines a triggering event of an activity state (start of an activity) as \textit{ViewReviews}. Then the response time from the previous example would be the time between the events \textit{ViewReviews} and \textit{EnterEvaluation} which seems more logical.

Clustering an actor’s tasks to sub-tasks, independent of interaction of another (slow) actor with the system, is the first step in the tasks to a navigation structure transformation process. The clusters are transformed to single NU, or groups of NU that are available in the single session. In order to define the partitioning of tasks to activity clusters we introduce the event order operator.

\begin{definition}[Event Order] Let \( F \) be a CS as it is specified in Definition 5.2 with the set of events \( E \). The relation \( \preceq \subseteq E \times E \) determines the order of events in the course of execution. For two subsequent events \( e_{\text{pre}}, e_{\text{succ}} \in E \) holds:

\[
e_{\text{pre}} \preceq e_{\text{succ}} \iff \exists 1 \leq i < n, a_i \in S, e_i \in E : a_i \xrightarrow{e_i} a_{i+1} \text{ where } e_1 = e_{\text{pre}}, e_{n-1} = e_{\text{succ}}
\]

It can be shown that the relation \( \preceq \) is a partial order of the set \( E \) (it is reflexive, antisymmetric and transitive). If for a particular CM two events are in the order \( e_1 \preceq e_2 \) then in any process instance of this model \( e_1 \) occurs before \( e_2 \).
Example 5.7 (Event Order) In the example collaboration model shown in Figure 5.3 the event EnterSubmission always occurs before the ViewSubmission which can be written as $\text{EnterSubmission} \preceq \text{ViewSubmission}$. In words: the activity EnterSubmission should be completed before the activity ViewSubmission is completed. This is so because the following relations hold in the CM:

$$\text{EnterSubmission} \xrightarrow{\text{EnterSubmission}[\text{submission.date} \leq \text{Conference.subDeadline}]} \text{StoreSubmission}$$

$$\text{StoreSubmission} \xrightarrow{\text{StoreReview}[\text{count(\text{review.inEval.of Paper})} < 2]} \rightarrow \text{ViewSubmission}$$

Note that the StoreSubmission event only occurs when the appropriate execution branch is chosen (based on the $[\text{submission.date} \leq \text{Conference.subDeadline}]$ guard condition in the decision), and the ViewSubmission can occur only when the composite event representing the merging of optional execution paths $\rightarrow \text{StoreReview}[\text{count(\text{review.inEval.of Paper})} < 2]$ occurs.

The order of events is enforced by the structure of CS expressed by CM. If events belonging to a single collaborative process instance occur in the wrong order, the process instance cannot be executed. The definition of the $\preceq$ relation is important for reasoning about the relationships between the CM and NM. It is essential for automatic construction of navigation infrastructure where the event order is ensued by proper linking of navigation clusters.

Definition 5.4 (Activity Cluster) Let $T_A(a) = \{s | \exists t : (s, t) \in \tau \land (a, t) \in \alpha\}$ be a set of activity states in tasks of a single actor “$a$” for a task model $< A, T, \alpha, t_S >$ and a task $CM < S, E, G, \rightarrow, S_0, S_t, T_s, T, \tau >$. An activity cluster is a set of activity states $C \subseteq S$ such that $\forall a, b, c \in C, \overrightarrow{a} \preceq \overrightarrow{b} \preceq \overrightarrow{c} \rightarrow a, b, c \in \{T_A(a) \cup \overrightarrow{T_A(\text{system})}\}$.

An activity cluster (AC) as described in Definition 5.4 is a set of activity states, where any three states in the cluster that must be performed subsequently must be performed by the same actor, or the system. This property ensures that there is no activity state in a cluster that must be performed by another actor, which means that performing activities in a cluster is independent on collaboration with other actors.

The goal of the partitioning step is to find the largest clusters possible based on the actor tasks. This facilitates restructuring of a CS into a navigation topology. The clustering procedure checks for every task the CS execution paths and adds all activity states that are not dependent on states from tasks of different actors. Figure 5.11 shows an example of the MakeSubmission task. The diagram shows the CS and the triggering events representing former states on the global process. The blocking event is the StoreEvaluation event that follows and depends on the states of the tasks of AssignReviewer, MakeReview and Evaluate tasks of different actors.
B. Generating Navigation Clusters

Navigation clusters (NC) are partitions of NM that specify a web user interface for activity clusters. The internal structure of NC mimics the sequences of activity states in the corresponding AC. However, there are possibly multiple (heuristic) strategies to achieve reasonable results for the automated generation of NC. One of the strategies (the default one currently used) is further partitioning of AC into units. These units are then transformed to navigation units using a one-to-one mapping. Concretely, the strategy is based on the following integration rules:

- Integration of subsequent information observation and information entry activities into one unit (slice) based on the common fact that an information entry usually follows after information observation (the user first must have some information to decide upon). It can be done for \( a_1 \in A_O, a_2 \in A_E \land a_1 \rightarrow a_2 \), where \( A_O \) is the set of all information observation activity states within the same task, and \( A_E \) is the set of all information entry activity states within the same task and

- Integration of subsequent information entry states into one unit (slice) can be done for \( \forall a_i \in A_E, a_i \rightarrow a_{i+1} \) and can be restricted by a maximal number of inputs.

These two rules represent a very simple but useful strategy for generating navigation units (slices). For example, the AssignReview task of PC member would be a single AC that renders the form showing paper info, list of reviewers, and allows entering of the paper into a reviewer assignment.

For the sake of conciseness we show only AC “Cluster 3” transformed to the navigation unit Cluster3 depicted in Figure 5.13 using the Hera AM graphical notation. The ViewSubmission activity is transformed to the Cluster3 NU (Figure 5.10) with the Paper.AssignRevForm unit presenting all new papers (submissions). They are stored as parameters of messages in the queue MQSubmission (see Figure 5.10). The abstract operator \( \text{get()} \) for retrieving messages from queues is implemented as a retrieval query of the slice Paper.AssignRev and a removal query triggered on the event LoadCluster3 triggered when the Cluster3 unit is instantiated (it removes retrieved messages from the queue). The paper title is used for retrieving the paper and author information from the domain database. The activity EnterAssignments is transformed into the Reviewer form allowing selection of reviewers. Assignments is an event triggering a service rule that puts all papers in the list that have not yet assigned reviewers, back to the MQSubmission queue and puts an Assignments message with the filled form instance as a parameter into the MQAssignments message queue (see Figure 5.10). It also triggers another processing step represented by subsequent queries (StoreAssignemnt, etc.).

5.6.3 Generation Procedure

The procedure for generating CMM and instantiating CDM contains all aforementioned steps. The procedure first creates the CMM by including ECA rules describing CM and
adding rules for message queueing and user management. The next step is partitioning tasks and generating AC. The navigation structure is put together from navigation clusters that are built from AC. Algorithm 1 describes the procedure for AM generation.

5.7 Conclusion

The design of WIS with intensive user communication and collaboration — as addressed in Research Question 4 in Chapter 1 — brings number of design issues that are difficult to express with the AM explained in the previous chapter. The current chapter presented a supporting lightweight process-model that facilitates specification of such systems. UML activity diagrams have been extended with a language allowing a detailed specification of activity states and subsequent automatic generation of AM. The generation procedure can use different heuristic methods for the generation of navigation clusters; the explained simple method is only an example of one of more possible options. Moreover, if actors are external systems, it is possible to generate facilities for calling Web services. Heuristic methods of AM generation (mainly its navigation part) and possibilities the generation process offers seem to be an interesting subject for further research.
Algorithm 1: Generation of AM

**input**: $TM = \langle A, T, \alpha, T_S \rangle$ and $CM = \langle S, E, G, \rightarrow, S_0, S_t, T_s, T, \tau \rangle$

**output**: $AM = \langle E_{AM}, R_{AM}, N_{AM}, \varepsilon \rangle, CDM$

initialization: $E_{AM} = \emptyset, R_{AM} = \emptyset, N_{AM} = \emptyset, \varepsilon = \emptyset, CDM \leftarrow \text{CreateCDM}();$

updating CM:

forall $s \in S$ do
    unfold composite states;
    if $IsComposite(s)$ then
        ReplaceState($s, \text{FindCM}(s)$);
    end
    remove direct communication;
    $s_{next} \leftarrow \text{GetNextState}(s, CM)$;
    $t \leftarrow \text{GetTask}(s, CM)$;
    $t_{next} \leftarrow \text{GetTask}(s_{next}, CM)$;
    if $t \neq t_{next} \land t_{next} \notin T$ then
        $Rearrange(s, s_{next}, T_s)$;
    end
end

add rules for all event;

forall $e \in E$ do
    if $IsComposite(e)$ then
        resolving composite events;
        AddEvent($AM, \text{ResolveEvent}(e)$);
    else
        creating message queues;
        if $IsMessage(a)$ then
            AddRules($AM, \text{CreateMsgQueueRules}(e)$);
            AddQueueSingleton($CDM, e$);
        end
        $E_{AM} \leftarrow E_{AM} + e$;
        AddRules($AM, \text{FindRules}(e)$)
    end
end

initializing AM;
AddRules($AM, \text{AddUserMagaementRules}(TM)$);
AddRules($AM, \text{GenerateNMSkeleton}()$);

partitioning task; $Clusters \leftarrow \text{MakeClusters}(CM)$;
building menus and navigation clusters;

forall $a \in A$ do
    $M \leftarrow \text{MakeMenu}(a)$;
    $N_{AM} \leftarrow N_{AM} + M$;
    forall $c \in Clusters \land (a, t) \in \alpha \land c \in t$ do
        AddMenuItem($M, c$);
        AddNavCluster($AM, \text{CreateNavCluster}(c)$);
    end
end
Chapter 6

Application Templates

Reuse is one of the major concepts of software design. Reuse of software specifications facilitated in the design process saves significant design effort and costs and makes the design more structured and transparent. There are different granularity levels of software specifications and thus also different granularity levels of reuse.

This chapter does not propose any new model but it aims at a reuse mechanism applied at the level of AM (Research Topic 5 described in Chapter 1). An application template is an AM that uses a virtual domain with a structure expressed by a template domain model. With a proper template parameter it can be deployed to a concrete domain. This parameter is a mapping from the virtual domain to the concrete one. The benefit of this approach is that complete functional parts of AM can be reused and only a mapping from a template’s virtual domain to a concrete domain needs to be specified. The focus of this chapter is in the specification of mappings, their representation and the process of AM template deployment.

Section 6.1 offers a brief overview of different types of reuse in software design for WIS. An overview of application templates is the subject of Section 6.2. The principles of application templates applied to the Hera method supported by a running example are outlined in Section 6.3. The mapping vocabulary and the application of mappings are briefly discussed in Section 6.4. Section 6.5 concludes this chapter.

6.1 Reuse in WIS design

The evolution of software engineering is to a large extent driven by the aim to improve the effectiveness of the design process and to minimize design efforts. One of the essential concepts that helps in this respects is reuse of software specifications. Reuse can rely on different techniques and it can be realized at different granularity levels. For instance, derived classes in an object-oriented software specification share (and reuse) the functionality of their parent classes, but this technique is rather different from reuse of class templates where the latter ones can be deployed in a concrete context determined by the template parameter(s) (typically also a class). Reuse can be applied to different levels of specification
from reuse of methods or method signatures to reuse of complex software components.

Due to the specific nature of the Web, extensions of traditional software design methods supporting the design of Web applications have been proposed. Practically all software design methods for the Web benefit from reuse techniques, mostly at finer levels of granularity (libraries of generic classes/concepts). However, flawless and practical reuse of navigation structure specification (often referred to as ”Web Patterns” or ”Web Design Patterns”) seems to be still a challenging problem. Unlike general design patterns using the Object-Oriented approach [Gamma et al., 1995], a usable set of Web patterns is still under discussion. A good overview of common Web patterns is presented in [Lyardet et al., 1999; Rossi et al., 2000]. The description can serve as a useful guideline for Web developers. Existing software libraries offer a wide variety of useful generic primitives that can be (re)used during the construction of Web applications, but they rarely contain larger navigation patterns. Since navigation models are usually closely coupled with concrete domains specified by Conceptual Models (CM), it is not an easy task to achieve domain portability.

Concrete existing methods for Web design benefit from the reuse concept in various ways. WebML specifies the navigation structure by means of different (predefined) types of units. The method allows easy composition of different units, but the (data) units are also associated with data they present or process. Object-oriented approaches like OO-H, UWE, or OOOWS show a solid approach supporting object-oriented reuse techniques such as class abstractions. The problem of domain portability of navigation models in object-oriented environment using the OOHDM method is discussed in [Schwabe et al., 1999]. Web design frameworks introduced there represent abstract navigation models or their parts that are isolated from concrete domains and can be instantiated to a concrete domain. The deployment process consists of deriving a concrete OOHDM model from a OOHDM-frame.

In this chapter we propose a reuse method that applies a template reuse technique to AM (or its functional meaningful parts) that makes the AM patterns domain portable. Unlike the Web patterns the AM templates (Template Application Model or TAM) are not only abstract models but components that can be deployed with little effort. This approach uses a mapping from a template domain model (TDM) specifying the structure of the data used within a template, to a concrete domain model. The mapping is a data (schema) integration model which implies that we can benefit from existing knowledge in this field of research. The process of deployment of such AT to a concrete domain can be automated by using an AT specification and an appropriate mapping to a concrete domain. We explain the basic concepts of the AT design and deployment on two examples using the Hera methodology (see also [Barna et al., 2005]).

6.2 Application Templates Overview

An Application Template (AT) is the specification of a meaningful part of the navigation structure and content manipulation rules that serves a (common) purpose and that can
be reused for a class of (similar) applications. An AT completely defines the application logic — the presentation logic and the business logic (functionality). There is no structural difference between AM and AT, the difference is only in the way in which they are deployed. An AT is in general a web application component that can be used in many similar applications and or used (multiple times) in a single application in different contexts. An example is a user selection device (virtual shopping basket) deployed in a single Web application for online shops once as a shopping basket for customers, while that same AT is also used for the selection of product categories of the customers interest providing input for a product search. An AT defines the presentation logic (navigation structure) and the business logic (functionality and eventual data updates) of (a part of) a web application, is domain data independent and can be deployed as a software executable (depends on the concrete design method used).

Every AT contains a template domain model (TDM) which is a minimal specification of data structures used in the template application model (TAM). When a concrete AT is being deployed, a mapping from TDM to a concrete domain is defined. The structure of concrete domain data is specified in a target domain model (DM). The minimality requirement for TDM is driven by practical considerations. Apart from other reasons minimality makes domain mappings simple. The process of AT design is the following:

- The specification of the AT purpose and interfaces,
- The specification of the AT template domain model (TDM),
- The specification of the AT presentation and business logic in a template application model (TAM).

Deployment of AT requires manual mapping of TDM to a concrete DM, which is often a relatively small effort compared to repeating the complete design of often used Web application patterns. In summary, the process of AT reuse and deployment consists of the following phases:

- Manual specification of the mapping from TDM to the target DM,
- Automated transformation of AT to (a part of) the target AM,
- Manual linking of the transformed AT to the target AM.

### 6.2.1 AT Interfaces

While embedding the template CMM means adding AT rules to the target AM, navigation interfaces must be bound to existing target AM elements. The navigation interfaces define links incoming into the AT navigation model from the target AM and links outgoing from the AT navigation model to the destination AM.
**Definition 6.1 (Application Template)** Let $TDM$ be a template domain model describing a data content and let $TAM$ be an application model describing the application logic of a WIS and based on $TDM$. Then an application template is the tuple $< TDM, TAM, \nu_{in}, \nu_{out} >$, where $\nu_{in}$ is the specification of navigation inputs and $\nu_{out}$ is the specification of navigation outputs.

The specification of navigation interfaces defines incoming and outgoing links and their parameters (they can be described as “connectors”). The parameter expressions determine the selection of instances represented by input/output interface links. All navigation interfaces use only data concepts from $TDM$ and are mapped together with all TAM in the deployment process. Figure 6.1 shows a transformed AT bound to the destination AM. Units represent abstract descriptions of navigation nodes connected by links.

Link destination concepts of input and output links in AT and AM must be correct. It is the responsibility of the designer to define destination NU in AM for outgoing links in the deployed AT and vice versa to define possible (at least one must exist) new links with destination NU in the AT.

### 6.2.2 AT Deployment

TDM defines only virtual data concepts, properties, and attributes that do not exist in a real domain, at least not in the form they are defined in TDM. During the deployment process the virtual domain is mapped into the target domain. By applying different mappings a single AT can be deployed in different domains.

A manually mapped AT is ready for the automated deployment process. The deployment process described further in this dissertation text replaces all references to virtual data concepts in AM with references to target domain concepts using an appropriate mapping. Figure 6.2 shows how a single AT (possibly taken from a library of AT) can be deployed within different AM specifying different applications within different domains. After AT are transformed into (parts of) AM for concrete domains they can be used as separate application specification or they can be embedded into existing application specifications. The deployed AT is a (part of) AM.
6.3 Application Templates in Hera

6.3.1 Target Domain Mapping

The TDM mapping step based on the manual TDM mapping is the core part of the AT deployment process. Such mapping is used for the automated translation of path expressions that define navigation units and rules during the TAM transformation process. A concrete TDM mapping expresses the relationship between the template domain and the target domain that defines semantic similarity of concepts, concept properties, or attributes. Naturally, not all data elements (concepts, properties and attributes) from the template domain model always have their direct counterparts in the target domain model. For this reason, for the construction of expressions that determine mapping targets, a set of operators applied to retrieved data sets is used. The mapping expressions are data retrieval queries using the operators that determine how the corresponding target domain data is retrieved.

Definition 6.2 (Mapping) A mapping is a set of pairs \( \{ < \pi_{\text{temp}}, Q_{\text{target}} > \} \), where \( \pi_{\text{temp}} \) is a path expressing a template concept, concept relationship, or attribute and \( Q_{\text{target}} \) is a query expression determining the corresponding concept, concept property, or attribute from the target domain.
It is useful to distinguish between mappings of concepts, mappings of concept properties and mapping of concept attributes. The reason is possibly different sets of queries used in different types of mappings and possibly different shapes of mapping expressions.

### A. Concept Mapping

Mapping concepts has the form \( \{< \pi_{\text{temp}}, Q_{\text{target}} >\} \), where \( \pi_{\text{term}} \) is a single identifier of a concept in TDM and \( Q_{\text{target}} \) is a query containing concepts from the target DM, eventually implementing set operators listed in Table 6.1. The operators in the Hera method use extended SeRQL queries. Concepts on which set operators are applied, must be of the same type (including sub-concepts). The most straightforward type of mapping is the direct mapping, where a single concept is mapped to a single concept. Another type of mapping is addition, representing a new concept that is not present in the TDM but is present in the DM. In fact, it is not a real mapping but the specification of the new concept. Union, intersection and difference are set operators that define extents of transformed concepts and they are used for defining more advanced mapping expressions.

<table>
<thead>
<tr>
<th>Short Notation</th>
<th>Description</th>
<th>SeRQL Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_{\text{temp}} \rightarrow C )</td>
<td>Direct mapping</td>
<td>SELECT A FROM {A}<a href="">rdf:type</a>{C}</td>
</tr>
<tr>
<td>( C_{\text{temp}} \rightarrow #Add(C) )</td>
<td>Addition</td>
<td>adding C to DM during deployment</td>
</tr>
<tr>
<td>( C_{\text{temp}} \rightarrow C_1 \bigcap C_2 )</td>
<td>Union</td>
<td></td>
</tr>
<tr>
<td>( C_{\text{temp}} \rightarrow C_1 \bigcup C_2 )</td>
<td>Intersection</td>
<td></td>
</tr>
<tr>
<td>( C_{\text{temp}} \rightarrow C_1 - C_2 )</td>
<td>Difference</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Operators for mapping concepts and their representation using SeRQL queries

### B. Concept Property Mapping

Mapping concept properties is similar to the concept mapping. In fact, mapped properties are range concepts of the properties, so this mapping is just a mapping of concepts that are accessed from their related concepts. Mappings in the form \( \{< \pi_{\text{prop}, \text{temp}}, Q_{\text{prop}, \text{target}} >\} \) contain property path \( \pi_{\text{prop}, \text{temp}} \) in the form \( \{C_1\} < P_1 >, \ldots, \{C_n\} < P_{\text{target}} > \) and \( Q_{\text{prop}, \text{target}} \) is a query traversing through the target domain properties and implementing operators listed in Table 6.2. The notation of the mappings in the table uses shortcuts, where \( P_{\text{temp}} \rightarrow P_1 \) is \( \{\text{Ctemp}\}<\text{Ptemp}\> \rightarrow \{\text{C}\}<\text{P1}\> \).
<table>
<thead>
<tr>
<th>Short Notation</th>
<th>Description</th>
<th>Query Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{temp} \rightarrow P$</td>
<td>Direct mapping</td>
<td>SELECT D FROM {C}&lt;P&gt;{D}</td>
</tr>
<tr>
<td>$P_{temp} \rightarrow P^{-1}$</td>
<td>Inverse property</td>
<td>SELECT D FROM {D}&lt;P&gt;{C}</td>
</tr>
<tr>
<td>$P_{temp} \rightarrow #Add(P)$</td>
<td>Addition</td>
<td>adding P to DM during deployment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SELECT D FROM {D}&lt;P&gt;{C}</td>
</tr>
<tr>
<td>$P_{temp} \rightarrow P_1 \cap P_2$</td>
<td>Union</td>
<td>SELECT D1 FROM {C1}&lt;P1&gt;{D1} UNION SELECT D2 FROM {C2}&lt;P2&gt;{D2}</td>
</tr>
<tr>
<td>$P_{temp} \rightarrow P_1 \cup P_2$</td>
<td>Intersection</td>
<td>SELECT D1 FROM {C1}&lt;P1&gt;{D1} INTERSECT SELECT D2 FROM {C2}&lt;P2&gt;{D2}</td>
</tr>
<tr>
<td>$P_{temp} \rightarrow P_1 - P_2$</td>
<td>Difference</td>
<td>SELECT D1 FROM {C2}&lt;P1&gt;{D1} MINUS SELECT D2 FROM {C2}&lt;P2&gt;{D2}</td>
</tr>
</tbody>
</table>

Table 6.2: Operators for mapping concept properties (ranges) and their representation using SeRQL queries

C. Attribute Mapping

Mapping attributes is slightly different than mapping concepts or properties. Attributes never represent sets and the mapping does not use the set operators. On the other hand there is a concatenation that can be used. Mappings \{<π_{attr,temp}, Q_{attr,target} >\} contain attribute $π_{attr,temp}$ in the shape \{C\}<ATarget> and $Q_{attr,target}$ is a query containing concatenation operators. Table 6.3 shows possible types of attribute mappings. An interesting operator is the variant operator $\lor$. It maps a template attribute to multiple possible attributes depending on concrete target domain sub-concepts. The target domain concept may have sub-concepts with different versions of the attribute. This operator is implemented using a union query. Note that in the last line of Table 6.3 the concepts $C_1$ and $C_2$ must have a common super-concept. For demonstration of the mapping and deployment of AT we present two examples. The first example describes a minimal conference submission system AT and its mapping to the presented submission domain on which mapping and possible mapping conflicts are demonstrated. The second example describes the guided tour AT and its embedding within an existing AM.

6.3.2 Publication Example

This example points out some possible mapping conflicts that can occur due to different structures of data in TDM and the target DM. The template application allows browsing through the publications of a research group members. For a clear demonstration of different conflicts and solutions we use a target domain shown in Figure 6.3 (shown together with TDM).
### A. Template Application Model

The navigation model of TAM shown in Figure 6.4 consists of three navigation units providing information about researchers in a group (`Group.Researchers`), information about all publications of a single researcher (`Researcher.Details`) and one unit allowing adding new papers (`AddingForm`). With the added rule `AddingPaper` the TAM defines simple browsing / adding papers application.

**RULE AddingPaper =**

```
{  
  ON AddPaper(R)  
  IF {true}  
    {  
      INSERT  
        {  
          {#new(tdm:Paper) AS P}  
          <#new(tdm:ptitle)>(Title);  
          <#new(tdm:url)>{URL};  
          <#new(tdm:published_at)>{Published};  
          <#new(tdm:author)>{R};  
          <#new(tdm:year)>{Year}  
        }  
      FROM  
        {  
          {form:AddingForm}<form:iptitle>{Title},  
          {form:AddingForm}<form:iurl>{URL},  
          {form:AddingForm}<form:ipub>{Published},  
          {form:AddingForm}<form:iyear>{Year}  
        }  
    }  
}
```
The toAddPaper link carries the information about the current instance of Researcher. This information appears as event parameter and the rule can use it for filling the value of the new paper. Table 6.4 shows a set of mappings for TDM concepts, concept properties and attributes.

B. Deployed Publication AT

The deployed AT shown in Figure 6.5 is a full AM and it represents the whole application. Mappings shown in the previous section have been applied on the original AM. The pquery query in the unit Person.Papers connects the child unit Paper.Info and it implements the mapping for the \{tdm:Researcher\}<tdm:papers> property:

```
SELECT X FROM \{X\}contributed_by\{P\} UNION SELECT X FROM \{X\}created_by\{P\}
```

In this query \(P\) is the instance of the Person concept given by the Person.Papers unit instance. Note the transformation of the original published_at attribute to the Publication.Publ sub-unit that has two specializations Journal.Publ and Proceedings.Publ. The navigation unit inheritance is a convincing way of implementing the variant (\(\lor\)) operator. Because of the two sub-concepts of the target domain Publication concept that appears in
the mapping of the template concept Paper the AddingPaper rule is transformed into the two rules:

RULE AddingPaper_ID1 =

{  
  ON AddPaper(R)
  IF { Type="Journal" }  
  {  
    QUERY
    {  
      SELECT{ Type }  
      FROM { {form:AddingForm}<form:ipubtype>{Type}}  
    }
    INSERT
    {  
      {#new(dm:Paper) AS P}  
        <#new(dm:title)>{Title};  
        <#new(dm:link)>{URL};  
        <#new(dm:published_as)>  
          {#new(dm:Journal})<#new(dm:jtitle)<Published>;  
          <#new(dm:author)}{R};  
          <#new(dm:year)}{Year}  
    }
    FROM
    {  
      {form:AddingForm}<form:iptitle>{Title},
    }  
  
}
Table 6.4: Mapping the guided tour TDM to the conference system DM

<table>
<thead>
<tr>
<th>Publication TDM Expressions</th>
<th>Target DM Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tdm:Group</code></td>
<td>#Add (singleton)</td>
</tr>
<tr>
<td><code>tdm:Researcher</code></td>
<td><code>dm:Person</code></td>
</tr>
<tr>
<td><code>tdm:Paper</code></td>
<td><code>dm:Paper</code></td>
</tr>
<tr>
<td><code>{tdm:Group}&lt;tdm:gname&gt;</code></td>
<td>#Add</td>
</tr>
<tr>
<td><code>{tdm:Researcher}&lt;tdm:name&gt;</code></td>
<td><code>{dm:Person}&lt;dm:fname&gt;</code> ⋄ <code>{dm:Person}&lt;dm:surname&gt;</code></td>
</tr>
<tr>
<td><code>{tdm:Researcher}&lt;tdm:email&gt;</code></td>
<td>-</td>
</tr>
<tr>
<td><code>{tdm:Researcher}&lt;tdm:papers&gt;</code></td>
<td><code>{dm:Member}&lt;dm:created_by&gt;^-1</code> ⋃ <code>{dm:Contributor}&lt;dm:contributed_by&gt;^-1</code></td>
</tr>
<tr>
<td><code>{tdm:Paper}&lt;ptitle&gt;</code></td>
<td><code>{dm:Paper}&lt;title&gt;</code></td>
</tr>
<tr>
<td><code>{tdm:Paper}&lt;published_at&gt;</code></td>
<td><code>{dm:Paper}&lt;published_as&gt;{Proceedings}&lt;ctitle&gt; ⋁ </code>{dm:Paper}&lt;published_as&gt;{Journal}&lt;jtitle&gt;`</td>
</tr>
<tr>
<td><code>{tdm:Paper}&lt;tdm:year&gt;</code></td>
<td><code>{dm:Paper}&lt;dm:year&gt;</code></td>
</tr>
<tr>
<td><code>{tdm:Paper}&lt;tdm:url&gt;</code></td>
<td><code>{dm:Paper}&lt;dm:link&gt;</code></td>
</tr>
</tbody>
</table>
Both rules are triggered by the form `Researcher.AddForm` submission (the `AddPaper` event). The update action of the `AddingPaper_ID1` rule is executed if the user selects the publication type “Journal” and the update action of the `AddingPaper_ID2` is executed if the user selects the publication type “Proceedings”. The input form `AddingForm` is generated automatically and its value determines which publication sub-type is going to be created. In general, when the transformation procedure finds an expression that is mapped using the variant operator $\lor$ the query is transformed to $n$ variants depending on the expression (see the `{tdm:Paper}<published_at>` mapping). In this case there are two variants for proceedings and journals. In order to provide the execution of the correct variant of the update query in the runtime, a form triggering the original update operation is detected and the input deciding the concept subtypes (in this example `dm:Journal` and `dm:Proceedings` sub-concepts of `dm:Paper` is added to the form.)
6.3 Application Templates in Hera

6.3.3 Mapping Conflicts

Even the simple example shows that the specification of mappings is not always trivial. Structural differences between data models of templates and target domains may cause conflicts that can be often resolved by using expressions explained in Section 6.3.1). Most of the possible conflicts are discussed and classified in [Sattler et al., 2003; Sheth and Kashyap, 1992]. Many of them can be expressed using the mapping operators presented in Section 6.3.1 and can be resolved automatically.

A. Data Representation Conflicts

In this case, types of corresponding attributes are not compatible. The solution is a simple type conversion, concretely the type of a conflicting TDM attribute is transformed (possible during the model transformation) to the data type of a corresponding CM attribute. Applied to our example the \{tdm:Paper\}<year> type Integer in the TAM is changed to String in the resulting AM.

B. Missing Concept

In this case, the missing concept is replaced with a single (virtual) constant concept, so that all its attributes are constants. In our example the designers of the concrete domain model aimed it just at a single (our) group. The group name will then be replaced with a constant string. The replacement by a constant is needed due to the fact that some top-level slices can be based on (non-existing) concepts. These slices are replaced by constant slices during the transformation process.

C. Missing Attribute

In this case, the missing attribute is omitted in the resulting concrete AM. An example is the email attribute of the Researcher.Details unit in TAM that does not appear in the resulting AM (see Figures 6.4 and 6.5).

D. Concept-Attribute and Attribute-Concept Conflicts

This problem occurs when a concept in TDM has a counterpart of the type attribute in DM or vice versa. In the publications example it does not occur, but the second reversed property-concept conflict appears in the mapping of \{tdm:Paper\}<tdm:published_at> to \{dm:Paper\}<dm:published_as>. This conflict is discussed in Section 6.3.3.

E. Generalization Conflict

This conflict can occur when a TDM concept is mapped to a DM concept with sub-concepts that have a different structure of properties and attributes. The example is the mapping the \{tdm:Paper\}<tdm:published_at> attribute that can be mapped into
The solution to this problem should be considered:

- **Transformation of navigation units.** In this case a navigation unit and corresponding specialized (inherited) navigation units, based on the general concept and its sub-concepts, are generated.

- **Transformation of data manipulation queries.** Especially with the creation of new instances of concepts with sub-concepts it is important to know the concrete type of the created concepts. With data manipulations associated with form processing a new input field is automatically generated that is used by users to decide a concrete type. For more details see the publication example in Section 6.3.2.

**F. Inverse Property**

This situation occurs when a TDM property does not have a direct counterpart in DM, but there is a DM property with inverse semantics. There is no direct example, but \{tdm:Researcher\}<tdm:papers> can be mapped to inversion of the union of \{dm:Paper\}<dm:created_by> ∪ \{dm:Paper\}<dm:contributed_by>.

**G. Attribute Concatenation Conflict**

This is the case when an attribute is mapped to a concatenation of multiple attributes. An example is the concatenation \{dm:Person\}<dm:fnamname> ⊗ \{dm:Person\}<dm:surname> for the \{tdm:Researcher\}<tdm:name>. The solution is replacing one TAM slice attribute in TAM by more attributes in the resulting AM.

### 6.3.4 Guided Tour Example

The guided tour is a widely used Web pattern. This example demonstrates the embedding of the mapped guided tour template into an existing AM.

**A. Guided Tour TDM**

The domain model of the guided tour AT is shown in Figure 6.6 at the top. It is a single concept called *Item* with optional literal property *description* that represents a descriptive label of concept instances. The concept *Pointer* always specifies the currently presented instance. *Pointer* is a singleton — it always has a single instance.
6.3 Application Templates in Hera

B. Guided Tour TAM

The Template application model defines the presentation logic (navigation structure) and the business logic of the template. Before explaining details of TAM we first discuss how the guided tour works in the context of Hera.

In order to provide sequential access to multiple instances of a domain concept (in TDM represented as the Item concept) we define the order in which the item instances are presented. This order is typically based on literal properties of the main concept (Item) or its related concept. In our example the order will be based on the alphabetical order of instances of the description property. Note that for the correct ordering (and navigation) the ordering values may not repeat. This feature can be ensured by adding an ordering attribute to all presented items.

The navigation model of the guided tour AT is simple and consists of the navigation unit presenting information about a current instance of Item (description). Moreover, it has a form with four navigation buttons allowing a navigation step forward and backwards and navigation to the first and last instance of Item. The NM part of TAM is shown in Figure 6.6.

An interesting part of the TAM is the CMM containing rules triggered by events FirstClicked, PrevClicked, NextClicked and LastClicked. These rules update the context data, storing the current state of the navigation of the guided tour. Concretely, the instance of the property current of the Pointer singleton is updated according to the type of next
navigation given by a button clicked by the user. The items presented using the guided tour template are ordered by an attribute of the Item concept (let us consider description for this purpose). The original state of the context data is established by the following rules triggered on an event appearing prior to the guided tour is used, for instance SessionStart.

RULE InitGT =
{
    ON SessionStart()
    IF {true}
    {
        INSERT
        {
            {#new(cm:Pointer)}<#new(cm:current)>{I}
        }
        WHERE
        {
            I= SELECT { MIN(D) }
            FROM { {dm:Item}<dm:description>{D} }
        }
    }
}

Assuming that the instance of Pointer does not exist this rule creates this singleton and an instance of the current property pointing to an instance of Item with minimal value of description (given the string type it is the alphabetically leftmost value). The following rule provides the update of the current property to a new value — the beginning of the order (which is the same as the initial one).

RULE First =
{
    ON FirstClicked()
    IF {true}
    {
        REMOVE { {P}<#remove(cm:current)>{} } FROM { {P}<rdf:type>{cm:Pointer} },
        INSERT { {cm:Pointer}<#new(cm:current)>{I} }
        WHERE
        {
            I= SELECT { MIN(D) }
            FROM { {dm:Item}<dm:description>{D} }
        }
    }
}

This rule does not create a new instance of the Pointer singleton, but only replaces the current property instance. The following rule provides moving of the pointer to the previous item (the item with smaller ordering value).
6.3 Application Templates in Hera

RULE Previous =
{
  ON PrevClicked()
  IF { E > 0 }
  {
    QUERY
    {
      SELECT { COUNT(I) AS E }
      FROM
      {
        {cm:Pointer}<cm:current>{dm:Item}<dm:description>{C},
        {dm:Item AS I}<dm:description>{D1}}
      WHERE { D1 < C }
    }
    REMOVE { {P}<#remove(cm:current)>{} }
    FROM { {P}<rdf:type>{cm:Pointer} },
    INSERT
    {
      {cm:Pointer}<#new(cm:current)>{I}
    }
    FROM
    {
      {I}<rdf:type>{dm:Item};
      <dm:description>{P}
    }
    WHERE
    { P =
      SELECT { MAX(D) }
      FROM { {dm:Item}<dm:description>{D} }
      WHERE { C>D }
    }
  }
}

The guarding condition of this rule checks whether there exists an item with a smaller ordering value than the current one. Analogically we can write the rules for navigation forward.

Because the described rules are not entirely trivial and because the guided tour is a pattern that is often required it is useful to reuse this application pattern as an AT. In the next section we explain how this AT can be used in the context of the conference review system for sequential access to e.g. selected papers.
C. Guided Tour Mapping and Deployment

The described guided tour AT can be used successfully in the context of larger applications. Consider the application described by the AM from Chapter 4. We shall modify it such that the content of the Review.ForReviewer NU will be presented as a guided tour (for the set of reviews by a single reviewer).

The mapping of an AT to a concrete domain is based on the specification of relationship between concepts and relationships in TDM and in the target DM. This mapping is used in the transformation of all path expressions in TAM referring to data concepts. The mapping for the guided tour AT to the conference system domain is shown in Table 6.5. There are two variants of mapping items (articulations). The first variant maps an expression (a concept, a property, or an attribute) to another expression and the second variant is just adding concepts (or properties/attributes) to the target domain model. Typically, the addition takes place when context data structures supporting AT functionality are not present in the target application. The process of the transformation to the target domain can be automated and is based on the rewriting data path expressions in TAM definition.

<table>
<thead>
<tr>
<th>GD TDM Expressions</th>
<th>CS DM Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tdm:Item</td>
<td>dm:Paper</td>
</tr>
<tr>
<td>{tdm:Item}<a href="">tdm:description</a></td>
<td>{dm:Paper}<a href="">dm:title</a></td>
</tr>
<tr>
<td>tdm:Pointer</td>
<td>Added</td>
</tr>
<tr>
<td>{tdm:Pointer}<a href="">tdm:current</a></td>
<td>Added</td>
</tr>
</tbody>
</table>

Table 6.5: Mapping the guided tour TDM to the conference system DM
After the automatic transformation of an AT it can be updated for its effective deployment. For instance, new attributes can be added to navigation units, new rules can be added, or some elements can be deleted. Figure 6.7 shows the transformed guided tour AT for the conference system with added attributes. The domain is extended with the Pointer concept and the current property. The resulting deployed guided tour within the target AM is shown in Figure 6.8. The guided tour here is placed into the Reviewer.Main navigation unit, but it can be also deployed as a separate unit. The input InputSet is associated with the relationship makes. The output is unused in this simple example.

6.4 Applying Mappings

6.4.1 Mapping Vocabulary

Mappings are represented in Hera analogically to other models using RDF/RDFS. The RDFS vocabulary of terms used in the mapping data is in Figure 6.9. A mapping contains a set of Articulations that define separate mapping specifications for a single concept, property, or attribute. An articulation contains a template data path and its counterpart expression composed from data paths, constants and operators. The PExpression concept represents a tree node (with leaves) of a classical expression tree. The from path expressions represent data elements from TDM and the to path expressions represent the target DM data elements which may use mapping operators. The mappings can be prepared by a tool described in Chapter 7. It stores the edited mappings in RDF files.
6.4.2 Mapping Procedure

The automated transformation procedure uses the prepared mappings and existing AT. The result is a deployed TAM that uses — instead of TDM concepts, properties and attributes — concepts, properties and attributes from the target DM. The transformation procedure processes all navigation units and all rules and transform them to the form corresponding to the target domain structure. The procedure is sketched in Algorithm 2. The inputs for the procedure are a TAM, an empty transformed TAM and the mapping specification. In summary, all path expressions (graph patterns) are searched in the TAM and concepts, properties and attributes are replaced by mapping expressions. This sometimes leads to complicated queries that are put into separate rules (e.g. in case of mapping expressions using the variant operators). The optimization of these queries can be the subject of future research. The procedure searches through all NU starting with $n_0$ and traverses through associated links, attributes, data compositions and rules.

6.5 Conclusion

In this chapter we focus on Research Topic 5 (Chapter 1) by explaining the principles of domain-portable application templates, their construction and their deployment. This technique can be used in a broader context and in this Chapter we demonstrated it in the context of the Hera method. Possible data conflicts occurring in the specification of a mapping from (virtual) template domains to target domains are well known and described in scientific literature. A simple set of mapping operators and techniques described in this chapter can help to resolve the majority of them. The application template technique is one of few techniques that allows reuse of fully functional domain portable application parts. The process of deployment is automated using a relatively simple algorithm. The automated application of a mapping to TAM can be further optimized. In addition, techniques for automation of the mapping specification (e.g. based on structural or semantic similarities) can be investigated in future.
Algorithm 2: AT Transformation

**input**: \( TAM = \langle E, R, n_0, N, \lambda, \epsilon \rangle, Mapping \)

**output**: \( TAM_{trans} = \langle E, R_{trans}, n_{0,trans}, N_{trans}, \lambda, \epsilon_{trans} \rangle \)

initialization: \( N_{trans} = \emptyset, R_{trans} = \emptyset \);

transforming NU;

forall \( n \in N \) do

\( o \leftarrow TransformConcept(\text{GetOwner}(n)) \);

\( L \leftarrow \text{GetLinks}(n) \);

forall \( l \in L \) do

\( TransformLink(l, Mapping) \);

end

\( Q \leftarrow \text{GetQueries}(n) \);

forall \( q \in Q \) do

\( TransformQuery(q, Mapping) \);

end

\( C \leftarrow \text{GetCompositions}(n) \);

forall \( c \in C \) do

\( TransformComposition(c, Mapping) \);

end

\( N_{trans} \leftarrow N_{trans} + CreateNU(o, L, Q, C) \);

end

transforming rules;

forall \( r \in R \) do

\( e \leftarrow \text{GetEvent}(E) \);

\( a \leftarrow \text{GetAction}(R) \);

\( c \leftarrow \text{GetCondiion}(R) \);

if \( \text{IsQuery}(a) \) then

\( q \leftarrow TransformQuery(a, Mapping) \);

\( a \leftarrow ReplaceAction(a, q) \);

end

if \( \text{HasRetrieve}(c) \) then

\( q \leftarrow TransformQuery(\text{GetRetrieve}(c), Mapping) \);

\( q_{C} \leftarrow ReplaceRetInCondition(c, q) \);

end

\( R_{trans} \leftarrow R_{trans} + CreateRule(e, c, a) \);

end
Chapter 7

Software Tools

The previous chapters explain the design principles and the design support techniques present in the Hera methodology. An important part of the study of software methods is the design and implementation of a set of tools allowing the investigated methods to be used and evaluated. This chapter addresses Research Topic 6 from Chapter 1. It summarizes the prototype software necessary to deploy and run a WIS designed using the extended Hera method. The core of these prototype tools is the Hera Web engine (Hera Presentation Generator) hosting Web applications specified by the set of Hera models. Furthermore, a set of supporting tools is presented.

The purpose and the structure of the Hera Presentation Generator is described in Section 7.1. Tools facilitating the design of models defining Web applications in Hera are explained in Section 7.2. The deployment of application templates requires proper mapping to target domains. A simple mapping editor described in Section 7.3 allows user-friendly viewing of two RDFS vocabularies and facilitates the specification of articulations (mapping) between them. The chapter is concluded in Section 7.4.

7.1 Hera Presentation Generator

The fundamental software tool is the engine providing runtime operation of WIS designed using Hera models. As it follows from the previous chapters the online page building process performed by this engine is triggered by user requests where pages are built on demand. This is both effective and necessary. It assures that the information used for Web page construction is up to date — it is always retrieved from data sources at the moment when a page is built. Even more important is the fact that due to the assumed dynamic nature of the data content it is very hard or impossible to generate all Web pages prior to their publication on the Web. The information content can be updated as an effect of user actions and the current state of the content should be always considered.

The Hera Web engine hosting Web applications is called Hera Presentation Generator (HPG). This engine is a Web server that reads Hera models, retrieves appropriate data, monitors user inputs and produces the resulting Web interface. Note that the data content
and a proper set of Hera models are sufficient to run a Web application. Besides the deployment information stored in the configuration files and the user repository explained later there is no need for additional programming and scripting. All Hera models except the optional presentation model are RDFS vocabularies. They use the vocabularies of modeling terms discussed in Chapter 3 and Chapter 4. Before explaining of how discussed modeling concepts are reflected in the HPG implementation we briefly explain HPG interfaces and how the system is configured to be operational.

### 7.1.1 HPG Interfaces and Configuration

It is obvious that HPG communicates with Web clients using the http protocol. Http requests are sent to HPG when a user follows a link or submits a form. Requests are discussed more in detail in Section 7.1.2.

Every Web application built using the Hera method starts with a user login. Although it is possible to log in as anonymous user, for many applications it is important to distinguish different types of users for the correct functioning of Web applications and for personalization purposes. Before the login information is processed HPG reads configuration information stored in the login page containing session data. The session data include model names, namespaces and engine configuration parameters like starting unit, use of dynamics, etc. Examples of the start page and login page are shown in Figure 7.1.

The HPG configuration for a concrete Web application is stored in the application start page. There the configuration parameters are implemented as invisible constant form inputs. The start page also contains the initial navigation context (see Chapter 4) that has a reference to the initial page (the \texttt{nuid} parameter) and its root concept instance ID (the \texttt{rootid} parameter). An excerpt of a login and configuration page source code for the publication example is:

```html
<form action="/Hera2WebApp/heraservlet" method="post">
...
<input type="hidden" name="amfile" value="Publications_AM.rdfs" >
<input type="hidden" name="cmifile" value="Publications_CMI.rdf" >
<input type="hidden" name="amnamespace" value="..." >
...
<input type="hidden" name="usedynamics" value="yes">  
...
<input type="hidden" name="nuid" value="Department.Initial">  
<input type="hidden" name="rootid" value="Department_ID1"> 
...
</form>
```
7.1 Hera Presentation Generator

Figure 7.1: HPG initial demo page (a) and login page for the publications demo (b)

7.1.2 Requests Processing Procedure

The main application (and only one from the users’ point of view) is simply receiving and responding to http requests. All functionality is performed within the generation of responses (http responses containing HTML page text). This functionality can include side-effects that are data content manipulations and possible invoking of external functionality.

Requests are organized in sessions that are created when the user logs in. Because multiple sessions can be run in parallel, requests are separate execution threads and all data access operations have been implemented in a thread-safe manner. Every request is processed through the following steps:

- **Decoding request type and parameters** and creating a new request processing thread of the determined type (form processing or link following). All following steps are executed in a single thread (for the current request) in a non-parallel manner.

- **Identifying types of events** that the request triggers followed by triggering appropriate CMM rules.

- **Retrieving data needed for the target page.** This is done by reading the proper NU specification.

- **Generation of target page in the end format (HTML).** This step in fact happens together with the previous step. The final layout of the constructed page can be determined by a presentation model (the current implementation uses a very simple default one).
Sessions are also generated in a dynamic way when a user logs in. All requests carry an identifier allowing to match it with registered (running) sessions. After this registration a session entry is stored in a session pool and it is removed from that pool when the session expires (the user logs off or closes his Web browser). A simplified response processing (which does not express its multi-threading nature) is shown in Figure 7.2.

![Activity diagram](image)

**Figure 7.2:** Activity diagram showing the collaboration of users and web applications using HPG

### 7.1.3 Decoding Requests

HPG requests are implementations of navigation events and they of course use the http request mechanism. HPG requests are divided into two types:

- *Link requests* caused by clicking on a link (anchor) and
- *Form submission requests* caused by clicking a form (submit) button triggering the form data processing followed by navigation (possibly to the same page with updated content).

The two types are distinguished (decoded) using the `METHOD` http parameter. All links (not only) generated by HPG use the `GET` method and all forms generated by HPG define the method type `POST` for their processing. Note that this method selection serves only for purposes of request decoding.

Besides standard http information (method type, request URI, host URI, etc.) HPG requests carry additional information essential for HPG operation. This information is
7.1 Hera Presentation Generator

Actually the extended representation of the navigation context discussed in Chapter 4. In order to simplify the access to the content of forms in case of form submission also the form content is sent as parameter. The additional information is transported in the form of standard http request parameters. The following HPG specific parameters are carried:

- **The identifier of the requested NU** is stored in the read-only (for AM) session variable `reqid` (session variables can be used in all queries within a single session); the identifier is the NU concept name in NM.

- **The identifier of the current NU** that belongs to the navigation context is stored in the read-only (for AM) session variable `nuid`.

- **The identifier of the root concept instance of the current NU** that also belongs to the navigation context is stored in the read-only session variable `rootid`.

- **The identifier of the triggering element** (link or form trigger) that determines the target NU is stored in the read-only session variable `triggerid`.

- **The form instance** present in case of form submission request. Its content is stored in a `form` session variable that is an RDF resources containing the data content of the submitted form.

The parameters stored as session variables can be used not only by HPG for construction of the response but they can be also used in queries used in NM and CMM.

7.1.4 Generating Events and CMM Rule Triggering

Triggering appropriate rules is essential for all content manipulations defined in CMM. Non-navigation events (external and temporal) are generated by SOAP listeners or timers and are treated asynchronously to request processing (rules they trigger run in a thread separated from all request threads). Navigation events are triggered by HPG requests and because of their synchronous nature within the request processing procedure they become rule execution commands rather than events. The order of rule execution commands differs slightly for form submissions and for links.

A. Link Rule Triggering Sequence

Consider $NU_D$ to be the destination NU of a link that triggered the link request. The sequence of rule triggering is simple and the appropriate rules are triggered in the following order:

1. Rules triggered by the event `BeforeLoad($NU_D$)` are executed,

2. After the $NU_D$ is instantiated and the response is generated and sent to the client Web browser `AfterLoad($NU_D$)` rules are executed.

The same holds for autolinks that determine $NU_D$ at runtime.
B. Form Submission Rule Triggering Sequence

The sequence of rule execution for form submission requests is similar to the sequence for link requests. The only difference is an extra (optional) event providing the form processing that is associated with a form trigger. Consider $T$ to be a trigger in the form that was submitted, $event(T)$ to be the event triggered for the form processing associated with $T$, $link(T)$ to be a (auto)link associated with $T$ and $NU_D$ to be the destination unit given by $link(T)$. The order of rule execution is the following:

1. Rules triggered by $event(T)$ are executed,
2. Rules triggered by the event $BeforeLoad(NU_D)$ are executed,
3. After the $NU_D$ is instantiated and the response is generated and sent to the client Web browser $AfterLoad(NU_D)$ rules are executed.

This sequence ensures that form processing takes place before the destination page is built, so the age can present updated information.

7.1.5 Generating Pages

The generation of pages consists of instantiation of NU and their transformation into the target format (e.g. HTML). This process is triggered by HPG (link or form) requests and it uses NM to specify the structure of the constructed NU and its actual data content. The data content is retrieved from the domain data content. NU instantiation also uses the current navigation context carried by the HPG request and available as session variables (see Section 7.1.3).

A. Processing NU Content

The NU instantiation process is based on modular structuring of NU. The process starts by evaluating the root query (possibly restricted by NU parameter) that calculates instance(s) of the root concept. The root concept instance is typically used for the calculation of NU child element instances. Children can be links, attributes, forms, or another NU. Note that the processing of the NU itself does not write anything to the resulting HTML stream, because the NU concept serves as a container for its child elements. The function implementing this process shown in Algorithm 3 in Appendix A can be recursively called multiple times (for child NU) when instantiating a NU. The elements (other than NU) are processed as follows:

- Links are instantiated by performing a link table query (possibly) restricted by a parameter passed from parent NU. The results are used for the calculation of anchor expressions that will be rendered. The last step is the generation of the end format text (HTML). The function providing link instantiation is shown in Algorithm 4. Note that autolinks set for dynamic adaptation of navigation paths are NM elements but they are not visual elements and thus they are not rendered.
• **Attributes** are instantiated by evaluating attribute expressions that can be seen as simple constrained queries that can operate only on the NU root concept. The evaluation of attribute expressions consists of parsing graph patterns contained in an expression using data statement functions similar to functions described in Section 7.1.6. When all needed information is retrieved operators in the attribute expression are applied and the resulting string is calculated and transformed into HTML. The function constructing attributes can be found in Algorithm 5.

• **Forms** instantiation is based on NU instantiation because the form concept is derived from the NU concept. Instantiation of form input fields is similar to attribute instantiation where input expressions are evaluated. In addition different types of inputs produce different HTML inputs (text input, selection input, etc.). Afterwards triggers are generated and transformed into HTML. The function building forms is shown in Algorithm 6.

### 7.1.6 Executing Retrieval and Content Manipulation Queries

Retrieval queries are essential for the specification of NM and update queries are essential for content manipulations specified in CMM. Although they represent different types of data access they share the notion of graph patterns. Details about the structure of queries and graph patterns can be found in Chapter 4 (retrieval queries) and Chapter 4 (data manipulation queries). Queries used in Hera AM are based on the SeRQL RDF query language. SeRQL however does not (yet) directly support data manipulations so data manipulation queries are parsed and processed by the HPG engine. The extension described in Chapter 4 presents content manipulation operators (CMO) \#new() and \#remove() that can be used in the **CONSTRUCT** clause (respectively the **REMOVE** clause) of data manipulation queries. Such queries are transformed into a sequence of Sesame API calls that manipulate the data model by adding or removing RDF statements (triples). Note that in this section we refer to data content as to data (graph) model to be consistent with expressions used in RDF and RDFS description.

For further explanation of query preprocessing and execution we briefly summarize the triple representation of RDF data. RDF [Beckett, 2004] is a data graph model that can be represented as a set of statements in the following form:

\[
\{S\}P\{O\}
\]

where \(S\) is a subject, \(P\) is a predicate and \(O\) is an object. This statement simply expresses the fact that the subject \(S\) has a property \(P\) with value \(O\). If subjects, properties and objects are terms from a data vocabulary (then subjects are concepts, properties are concept properties and objects are either concepts or literal value types) sets of triples can express graph patterns defined in Chapter 4. If concepts, properties and values are bound to variables and if these variables are used in constraining expression we can use them for data retrieval. Note the substantial semantic difference between *data statements* containing resources, resource properties and data values (RDF) that describe manipulated data.
graphs and graph pattern statements (or GP statements) that contain concepts, properties and literals that can describe vocabularies and are also used for internal representation of graph patterns (GP) in queries.

To support the explanation of query execution and to abstract from a concrete API we introduce extended statements, their notation and a set of operations providing partial query execution. The triple statements are GP statements that may contain CMO and that are used for building extended graph patterns. An extended statement is written as:

$$\{\overline{S}\} \overline{P}\{\overline{O}\}$$

$\overline{S}$ is either a concept (with or without binding variable), or a CMO creating (removing) a given concept (again with or without binding variable). $\overline{P}$ is either a property (with or without binding variable) or a CMO creating (removing) a given property (also with or without binding variable). $\overline{O}$ is either a concept or a value (possibly bound by a variable) or it is a CMO creating (removing) a given concept (possibly bound by a variable). Usage of a CMO within an extended statement is restricted by rules explained later. The following operators represent procedures used in HPG (and in algorithms at the end of this chapter) to execute content manipulation queries:

- **TableQuery**($D, Q$) performs a table retrieval query (selection query) on the domain $D$ and returns a set of concept instances (resources) and/or literal values in a form of table specified in the SELECT (projection) clause,

- **GraphQuery**($D, Q$) performs a graph retrieval query on the domain $D$ (in SeRQL it is called construction query but this term is overloaded in Hera such that construction queries can be also content manipulation queries) and returns a set of data statements describing the retrieved data graph,

- **PrepareExtentQuery**($\overline{C}$) transforms a content manipulation query $\overline{C}$ into a table (selection) retrieval query that calculates instances of concepts bound and restricted in $\overline{C}$ (FROM and WHERE clauses).

- **CreateURI**($D, c$) creates new instances of the concept $c$ with a new URI in the domain $D$,

- **Find**($D, s_{GP}, ExtTable$) retrieves all data statements from the domain $D$ that can be matched with the GP statement $s_{GP}$ that uses instances from the table $ExtTable$.

- **AddStatements**($D, S$) adds a set of data statements $S$ to the domain $D$.

- **MakeConstructStatements**($s_{GP}, ExtTable$) is used for construction data manipulation queries and it generates from the GP statement $s_{GP}$, a set of data statements representing the data sub-model represented by $s_{GP}$; it uses a set of instances (extents) $ExtTable$ returned by the extent query,

In the following paragraphs we discuss the execution of both types of content manipulation queries.
A. Executing Retrieval Queries

Data retrieval queries are part of the SeRQL specification and they do not require any additional analysis or an extra execution provided by HPG. The retrieved results are however temporarily stored in variables carrying the same identifiers as the used query binding variables and they can be used in the scope of CMM and NM specifications as follows:

- **In NM specification** the variables can be used (read) within the element (NU, form, link, attribute) that contains the query specification or in its child element,

- **In CMM specification** the variables can be used within the rule where the query is defined (used), for example if it is a (ECA) condition query the variables can be read in the action part.

As mentioned before, data retrieval queries can be table queries and graph queries, where queries of the first type return tables of values (similar to SQL queries) and graph queries return a data graph internally represented as a set of data statements (triples). Queries are written in the form shown in Figure 7.3 where *List* is a list of concepts or literal values of which types instances (values) are retrieved by a table query. A *GraphPattern* in the optional FROM clause is a GP specifying relationships between concepts relevant for the query and a *RestrictionExpression* in the optional WHERE clause is an expression providing additional constraints for the query. A *GraphPattern* expression in CONSTRUCT clause of graph queries is a GP describing the retrieved data graph. specifying

```sql
SELECT List FROM GraphPattern WHERE RestrictionExpression

CONSTRUCT GraphPattern FROM GraphPattern WHERE RestrictionExpression
```

Figure 7.3: Retrieval table query (left) and retrieval graph query (right)

The execution of a retrieval query is expressed by Algorithm 7. It is a simple sequence of operations that decodes the type of query, performs it and stores the result in the temporal HPG model that can be used in the actual query scope.

B. Executing Content Manipulation Construction Queries

Construction queries add data statements to a data model. The extended GP used in this type of queries can contain only #new() CMO. Construction queries use a syntax similar to graph retrieval queries where the retrieval GP in the CONSTRUCT clause is replaced with an extended GP (containing CMO):

```sql
CONSTRUCT ExtendedGraphPattern FROM GraphPattern WHERE RestrictionExpression
```
Executing this type of queries requires HPG to perform a few more operations than for retrieval queries. First we introduce three constraining rules for writing construction (and removal) queries that allow to execute them in a unambiguous and implementable way.

- The extended GP in the CONSTRUCT (REMOVE) clause can contain only paths where every statement (triple) contains at least one CMO. This rule assures that the first clause specifies only a data graph that is added to (removed from) the content and it facilitates query processing.

- Variables in the CONSTRUCT (REMOVE) clause GP must be restricted by FORM and WHERE clauses such that they can be evaluated and used for the creation of a data graph. This is simply provided by creating a table query retrieving the variables where the FROM and WHERE clauses are simply taken from the original construction query (it is provided by the PrepareExtentQuery(Traversal) function) and checking its correctness (executing) using the Sesame database engine.

- GP statements in the CONSTRUCT clause can have the following forms in respect to using the \#new() operator:

  - \{\#new(S)\}P\{O\} creates a single instance of S for possibly multiple instances of O given by the query (one-to-many construct).
  - \{\#new(S)\}\#new(P)\{O\} creates possible multiple instances of S for all instances of O given by the query (multiple one-to-one construct).
  - \{S\}P\{\#new(O)\} creates a single instance of O for possibly multiple instances of S given by the query (many-to-one construct).
  - \{S\}\#new(P)\{\#new(O)\} creates possibly multiple instances of O for all instances of S given by the query (multiple one-to-one construct).
  - \{S\}\#new(P)\{O\}, creates a new relationship between S and O instance. This statement can be used only when the query gives exactly one instance of S and one instance of O.
  - \{\#new(S)\}P\{\#new(O)\} creates one instance of S and one instance of O (one-to-one construct). The same effect have statements written in the form \{\#new(S)\}\#new(P)\{\#new(O)\}

These rules are simple and do not considerably restrict the use of CONSTRUCT queries. If the \#new(c) operator is applied to a concept the CreateURI(c) method that prepares a new URI for a new resource is called. The \#new(o) operator applied to objects of a literal type must provide a concrete data value. All newly created resources (concepts) can be bound to variables and used in FROM and WHERE clauses. Now we explain which statements are generated for the mentioned forms of GP triples that exactly describes the implementation of the MakeConstructStatements(sGP, ExtTable) function introduced in Section 7.1.6:
• \{\#new(S)\}P\{O\} is transformed to the function call \( c = CreateURI(S) \) and the following data statements:

1. \{c\} rdf:type \{S\}
2. \{c\} rdf:type \{SSuper\}
3. \{c\} P \{EO\}

The second statement provides model completeness by associating the resource \( c \) with all super-classes of the concept \( S \) using the \texttt{rdf:type} property. In other words it states that the type of the resource \( c \) is not only the concept \( S \) but also all its super-classes represented here by the set \( SSuper \). Note that the second statement then in fact represents a set of statements — one for every item in \( SSuper \). The third statement (possibly) represents also a set of statements for all considered instances of objects calculated by the extent query (or for objects of the literal type it is a single statement and a single data value).

• \{\#new(S)\}\#new(P)\{O\} is transformed into multiple function calls of \( CreateURI(S) \) that create a set of \( S \) instances \( ES \) with the same size as the size of the extent set \( EO \). The following statements are generated:

1. \{ES\} rdf:type \{S\}
2. \{ES\} rdf:type \{SSuper\}
3. \{ES\} P \{EO\}

All three statements represent sets of statements for every instance in \( ES \) (times every superclass for the second statement). The third rule makes a relationship of the type \( P \) for every new instance in \( ES \) with one distinct existing instance in \( EO \) (one-to-one relationship).

• \{S\}P\{\#new(O)\} is transformed to the function call \( c = CreateURI(O) \) that creates a new instance of \( O \) (when the object is a concept) or a new data value provided as parameter of \#new\( (\text{value}) \). The following statements are generated:

1. \{c\} rdf:type \{O\} only for objects of type concept
2. \{c\} rdf:type \{OSuper\} only for objects of type concept
3. \{ES\} P \{c\}

Similarly to the first case the second statement in fact represents a set of statements — one for every item in \( OSuper \). These triples ensure model completeness regarding the \texttt{rdf:type} property. The third statement can, for non-literal objects, be a set of statements due to possibly multiple instances in the set \( ES \).
• \{S\}\_new(P)\{\_new(O)\} is transformed into multiple function calls of CreateURI(O) that create a set of O instances (for literals a set of the same values provided as \_new(O) parameter) \(E_O\) with the same size as that of the extent set \(E_S\). The following statements are generated:

1. \(\{E_O\} \text{rdf:type}\ \{O\}\)
2. \(\{E_O\} \text{rdf:type}\ \{O_{Super}\}\)
3. \(\{E_S\} \ P\ \{E_O\}\)

All three statements again represent sets of statements for every instance (or value for literal objects) in \(E_O\) (times every super-class of \(O\) in the second statement). The third rule makes a relationship of the type \(P\) for every new instance in \(E_S\) with one distinct existing instance (value) in \(E_O\) (one-to-one relationship).

• \{\_new(S)\}_P\{\_new(O)\} is allowed only when sets \(E_S\) and \(E_O\) have size one and is transformed into the following single statement:

1. \(\{E_S\} \ P\ \{E_O\}\)

Due to restricted sizes of sets \(E_S\) and \(E_O\) this statement is indeed always a single statement.

• \{\_new(S)\}_P\{\_new(O)\} is transformed into function calls \(s = CreateURI(S)\) and \(o = CreateURI(o)\) and the following statements are generated:

1. \(\{s\} \text{rdf:type}\ \{S\}\)
2. \(\{s\} \text{rdf:type}\ \{S_{Super}\}\)
3. \(\{o\} \text{rdf:type}\ \{O\}\)
4. \(\{o\} \text{rdf:type}\ \{O_{Super}\}\)
5. \(\{s\} \ P\ \{o\}\)

The second and fourth statements may represent sets of statements for all superclasses of \(S\) (\(O\) respectively).

The whole execution of construction queries consists of building and execution of the extent table query that provides sets of instances (or single values) of concepts (literals) related to newly added resources/values, transformation of the query into a set of data statements by parsing the CONSTRUCT GP and adding generated statements into the original data model. This process is performed in a transactional fashion and if some of the steps fail the original data model (content) is not changed. For this purpose statements are added into a temporal model and only when all statements are successfully generated the temporal model is included in the original one. The process is described in Algorithm 8.
C. Executing Content Manipulation Removal Queries

Although removal queries remove statements from the data content and do the opposite of construction queries, their processing uses many similar principles. Nevertheless, although the processing of removal queries is similar to the processing of construction queries it features some differences discussed in this paragraph. The syntax of removal queries is:

\[
\text{REMOVE ExtendedGraphPattern} \\
\text{FROM GraphPattern} \\
\text{WHERE RestrictionExpression}
\]

Besides the operators introduced in the previous paragraph of which some are also used for the removal query processing the following operators are needed:

- \(\text{RemoveStatements}(D, S)\) removes a set of data statements \(S\) from the domain \(D\),
- \(\text{MakeRemoveStatements}(s, \text{ExtTable})\) is used for removal data manipulation queries and it generates from the GP statement \(s\) a set of data statements representing the data sub-model represented by \(s\); it uses a set of instances (extents) \(\text{ExtTable}\) returned by the extent query.

The main difference between construction and removal query processing is that removal queries do not create new resources, thus the generation of data statements in the query processing differs from the one for construction queries, and the generated data statements are not added but removed from the domain using the \(\text{RemoveStatements}(D, S)\). The rules constraining writing removal queries are identical to those described in the previous paragraph except the third rule restricting allowed forms of statements in GP in the REMOVE clause in respect to the \(#\text{remove}(c)\) operator. For removal queries the allowed forms are as follows:

- \{\#\text{remove}(S)\}P\{O\}\) (can be written also as \{\#\text{remove}(S)\}\#\text{remove}(P)\{O\}\) removes all instances of \(S\) having a relationship of type \(P\) with the extent (given by the query) of \(O\)
- \{\(S\}\#\text{remove}(P)\{O\}\) removes all relationships of type \(P\) between all possible combinations from extent sets (given by the query) for \(S\) and \(O\)
- \{\(S\)P\#\text{remove}(O)\}\) (can be written also as \{\(S\)\#\text{remove}(P)\#\text{remove}(O)\}) removes all instances (values for literals) of \(O\) for all instances of in the extent of \(S\) that have property \(P\) with object \(O\)

The remaining combinations \{\#\text{remove}(S)\)#\text{remove}(P)#\text{remove}(O)\} and \{\#\text{remove}(S)\}P\{#\text{remove}(O)\}\) representing the same statement would remove all instances of all its domain and range concepts and therefore it is not supported. These forms of statement are again transformed into different data statements (provided by the implementation of the \text{MakeRemoveStatements}(s, \text{ExtTable})\) function):
• For \( \{\#\text{remove}(S)\}P\{O\} \) the following type statements are generated:

1. \( \{E_S\} P \{E_O\} \)
2. \( \{E_S\} \text{rdf:}\text{type} \{S\} \)
3. \( \{E_S\} \text{rdf:}\text{type} \{S_{\text{Super}}\} \)

This set of data statements is based on the extent set \( E_S \) of \( S \).

• For \( \{S\}\#\text{remove}(P)\{O\} \) the following type statements are generated:

1. \( \{E_S\} P \{E_O\} \)
2. \( \{E_S\} \text{rdf:}\text{type} \{S\} \)
3. \( \{E_S\} \text{rdf:}\text{type} \{S_{\text{Super}}\} \)
4. \( \{E_O\} \text{rdf:}\text{type} \{O\} \) only for non-literal objects
5. \( \{E_O\} \text{rdf:}\text{type} \{O_{\text{Super}}\} \) only for non-literal objects

This set of data statements is based on the extent sets \( E_S \) (\( E_O \) if non-literal) of \( S \) (\( O \) if non-literal).

• \( \{S\}P\{\#\text{remove}(O)\} \) the following type statements are generated:

1. \( \{E_S\} P \{E_O\} \)
2. \( \{E_O\} \text{rdf:}\text{type} \{O\} \) only for non-literal objects
3. \( \{E_O\} \text{rdf:}\text{type} \{O_{\text{Super}}\} \) only for non-literal objects

This set of data statements is based on the extent sets \( E_O \) of \( O \) instances if the object is not literal.

The process of removal query execution consists of the creation of an extent table query, generation of the extent table, generation of data statements using the REMOVE clause GP and the extent table and removing generated statements from the original data model (if found). The extent table query retrieves also instances of concepts (values) on which the \#\text{remove}() operator is applied regardless whether they are bound or not. The reason is that the operator is applied by default to all instances when not restricted. This typically leads to substantial lower performance than in the execution of construction queries. The execution of a whole query is performed in a transactional fashion and it is summarized in the Algorithm 9.

### 7.1.7 Timers and External Communication

Timers and external (SOAP) listeners cause asynchronous system events that are treated in separate threads with thread-safe access to models and data content.
A. Timers

The implementation of timers is straightforward and it uses dedicated Java library classes for this purpose.

B. Invoking External Calls

The implementation of external call invocation is very simple and it requires the designer to specify all needed parameters that will allow to locate and invoke the desired Web Service as it is described in Chapter 4 (service and operation name and description of parameter messages). The specification of the return message that is the part of the call specification is however used in a SOAP listener (summarized in the next paragraph) that triggers a specified content manipulation query that processes the result of the call. The external call invocation consists of a simple transformation of the HPG call specification into SOAP messages and of sending prepared messages.

C. External Listeners

External listeners use the WSDL specification that exposes the HPG Web Service to the Web. It is constructed from AM and it contains:

- Methods defined in AM as a part of functionality exposed to external systems,
- Return messages for external functions that also trigger CMM rules.

When a listener receives a remote call (SOAP message) it matches its parameters with a proper event parameters and triggers the appropriate rule(s).

7.1.8 Implementation Remarks

The HPG system (the version that did not allow dynamic building of pages) was originally implemented as a set of XSL templates that processed data and accessed RDFS models at the level of XML syntax. This prototype proved the value of Hera modeling concepts but this implementation has clear disadvantages not only because of complicated implementation, but also because RDF and RDFS XML serialization allows multiple ways to express the same statements, which caused severe restrictions on the format of model and data.

The current implementation of the HPG engine uses the dedicated RDF data repository Sesame that reads any XML serialization of RDF and RDFS without any problem. HPG is now a Java servlet that processes http requests in a standard way. The simplified static class structure of the HPG servlet is shown in Figure 7.4. The class that processes the http request coming from the client side (browsers) is the singleton HPGServlet specialization of Servlet. RequestHandler encodes http requests (extracts event types and event parameters) and manages Session instances. EventHandler further dispatches events and provides instantiation of pages through a specialization of PageBuilder and execution of rules by RuleExecutor. The ExternalListener subclasses have callback ability to notify
*EventHandler* about incoming external calls. *ModelFactory* manages the creation of model readers and distributes all retrieval and data update requests. The *SesameSAILInterface* class is a specialization of *QueryProcessor* that provides the interface to the Sesame engine.

Figure 7.5 shows the request-response scenario as a message sequence chart referring to instances of HPG classes. The request is by default processed by calling the *doGet()* method from the web server. It is known that the selection of the processing method (*doGet()* or another) can be changed inside the HTML (or another hypertext format) page where the submitted form or activated link is defined. The request caused by a user action is passed onto the web server that calls the method. For this scenario, the AM model contains a rule triggered by the *BeforeLoad* associated with the requested page. The specification of the event and the rule(s) it triggers is found in AM through the message call *GetElement()* applied to an instance of the *ModelFactory* that is redistributed further to the data repository interface *SesameSAILInterface*. The rule is executed and possibly a content manipulation query is executed (triggered by *update()* call). In the next step a page is built as described before via *CreateNU()*/*CreateAttribute()*/*CreateLink()* and *CreateForm()* calls. At the end the response is returned to the web server that invoked the *doGet()* method.

### 7.2 Support for the Construction of Design Models

Besides relevant domain data content stored in an RDF repository a domain model (possibly including a context model) and an application model are necessary models needed for the deployment of a WIS. A presentation model is important also but it is out of the scope of this dissertation and a default one is used. Both DM and AM are regular RDFS vocabularies. Whereas DM does not have any specific generic vocabulary (besides the default RDFS) that restricts its structure, AM is Hera specific and has a vocabulary described in Chapter 3 and Chapter 4.

To help the designers to build and check these models we have developed a DM Builder and an AM Builder. The first versions of these tools were extensions of the Microsoft Visio drawing tool, implemented in Visual Basic for Visio. The DM Builder can be used for visual specification of arbitrary RDFS vocabularies and it is shown in Figure 7.6. Ovals represent RDFS concepts and rectangles represent literal attributes. Arrows represent concept properties, literal properties and dashed arrows *subClass* and *subProperty* properties. The DM Builder supports the specification of a cardinality property that is Hera specific. However, any available RDFS tool can be used for the visual construction of DM.

The AM Builder is an important part of the Hera design support. The AM Hera graphical notation differs from the standard RDFS graphical notation (or any other notation known to us). In addition, AM depends on the corresponding DM so it is very useful to have a tool supporting its design and checking its consistency with DM. After startup the tool prompts for loading a DM. Based on the DM structure it offers (constraints) options of the created elements. Figure 7.7 shows the AM builder environment while it is building a basic AM for the conference domain.
7.3 Support for Deployment of Application Templates

The most demanding tasks within the design and deployment of application templates are their specification and the specification of mappings for target domains. The aforementioned DM and AM Builders (or Hera Studio) can be used for the specification of AT and the AT Mapper can be used for the specification of the mappings. This user interface is based on the SeRQL query builder interface, designed for integration purposes and described in [Vdovjak et al., 2003] but it offers easier construction of articulations. Figure 7.8 shows the articulation building window of the tool. The tool allows to open two RDFS vocabularies and to edit sets of articulations. The vocabularies are viewed as tree structures with the hierarchy of concepts given by the concept specialization. Concept properties and attributes are also displayed. Both tree views (source representing the template DM and target representing target the DM) allow to build paths by double-clicking on concept properties or attributes. The navigation to possible concept domains that have been double-clicked is automatic and the building of paths is extremely easy. The target tree allows for the construction of path expressions described in Chapter 6. Visually specified mappings are stored in the format described in Chapter 6 and they can be directly used for transforming TAM for the target domain.

7.4 Conclusion

Software tools enable the deployment and operation of WIS built using techniques described in this dissertation. This section addresses Research Question 6 in Chapter 1. It describes software tools used for experimenting and evaluating the Hera method. The most important is the HPG engine that allows runtime generation of web pages based on the Hera models. We explain the implementation of application modeling in HPG concepts discussed in Chapters 3 and 4. The design support tools are very important for rapid and effective development of WIS using Hera models. These tools are at present being integrated into one development environment (Hera Studio). Most of the used development technologies for all software tools are based on Java, Java Servlets, J2EE and the Sesame API. New technologies such as .Net (at version 3) and Windows Presentation Foundation may in the future facilitate the implementation of new versions of the Hera engine and supporting tools.
Algorithm 3: NU generation function

CreateNU(D, NM, nu, Session, OutHTMLStream);

begin
    including ← true;
    if NotIsTopLevel(NM, nu) then
        C ← FindInclusionCondition(NM, nu);
    if IsOK(C) then
        including ← EvaluateCondition(D, C, Session);
    if including = true then
        Q = FindRootQuery(NM, nu);
        QRoot ← ReplaceSessionVariables(Q, Session);
        TempModel ← PerformQuery(D, QRoot);
        Session.AddModel(TempModel);
        Elems ← FindTypeStatements(NM, nu, "av:Link|av:Attribute|av:Form");
        for all s ∈ Elems do
            if IsSubclassOfNU(s.Object()) then
                CreateNU(D, NM, s.Subject(), Session, OutHTMLStream);
            if IsSubclassOfLink(s.Object()) then
                CreateLink(D, NM, s.Subject(), Session, OutHTMLStream);
            if IsSubclassOfAttribute(s.Object()) then
                CreateAttribute(D, NM, s.Subject(), Session, OutHTMLStream);
            if IsSubclassOfForm(s.Object()) then
                CreateForm(D, NM, Session, s.Subject(), OutHTMLStream);
        Session.RemoveModel(TempModel);
end
### Algorithm 4: Link generation function

`CreateLink(D, NM, link, Session, OutHTMLStream);`

```
begin
  including ← true;
  C ← FindInclusionCondition(NM, link);
  if IsOK(C) then
    including ← EvaluateCondition(D, C, Session);
  end
  if (including = true) And Not(IsAutoLink(link)) then
    Q ← FindLinkQuery(NM, link);
    Qₐ ← ReplaceSessionVariables(Q, Session);
    LinkTable ← TableQuery(D, Qₐ);
    AnchorExpression ← FindAnchorExpression(NM, link);
    forall row ∈ LinkTable do
      Anchor ← CreateAnchor(AnchorExpression, row);
      OutHTMLStream.Add(Anchor.GenerateHTML());
    end
  end
end
```

### Algorithm 5: Attribute generation function

`CreateAttribute(D, NM, attribute, Session, OutHTMLStream);`

```
begin
  including ← true;
  C ← FindInclusionCondition(NM, attribute);
  if IsOK(C) then
    including ← EvaluateCondition(D, C, Session);
  end
  if including = true then
    AExpression ← FindAExpression(NM, attribute);
    AttributeGP ← ParseAExpression(AExpression);
    Attribute ← Evaluate(D, AttributeGP, Session);
    OutHTMLStream.Add(Attribute.GenerateHTML());
  end
end
```
**Algorithm 6**: Form generation function

```plaintext
CreateForm(D, NM, form, Session, OutHTMLStream);
begin
    including ← true;
    C ← FindInclusionCondition(NM, form);
    if IsOK(C) then
        including ← EvaluateCondition(D, C, Session);
    end
    if including = true then
        OutHTMLStream.Add(FormBegin());
        CreateNU(D, NM, form, Session, OutHTMLStream);
        Inputs ← FindTypeStatements(NM, form, "av:Input");
        forall s ∈ Inputs do
            AEExpression ← FindAExpression(NM, s.Subject());
            AttributeGP ← ParseAExpression(AEExpression);
            Input ← EvaluateInput(D, AttributeGP, Session);
            OutHTMLStream.Add(Input.GenerateHTML());
        end
        Triggers ← FindTypeStatements(NM, form, "av:Trigger");
        forall s ∈ Triggers do
            Trigger ← FindTrigger(NM, s.Subject());
            OutHTMLStream.Add(Trigger.GenerateHTML());
        end
        OutHTMLStream.Add(FormEnd());
    end
end
```

**Algorithm 7**: Performing a retrieval query

```plaintext
input : D, Q
output: TempModel = {TempTable|TempGraph}
initialization: TempModel ← ∅;
if LookFirstClause(Q) = 'SELECT' then
    TempTable ← TableQuery(D, Q);
else
    TempGraph ← GraphQuery(D, Q);
end
```
Algorithm 8: Performing a content manipulation construction query

\textbf{input} : \( D, Q \)

\textbf{initialization:} \( \text{TempModel} \leftarrow \emptyset, Q_E = \text{PrepareExtentQuery}(Q), ok = true; \)
\( \text{ExtTable} \leftarrow \text{TableQuery}(D, Q_E); \)
if \( \text{IsOK(ExtTable)} \) then
  \begin{align*}
    &\text{forall } s \in \text{ConstructGP}(Q) \text{ do} \\
    &\quad \text{if } ok = true \text{ then} \\
    &\quad\quad \text{Statements} \leftarrow \text{MakeConstructStatements}(s, \text{ExtTable}); \\
    &\quad\quad \text{if } \text{IsOK(Statements)} \text{ then} \\
    &\quad\quad\quad \text{AddStatements(\text{TempModel}, \text{Statements})}; \\
    &\quad\quad\text{else} \\
    &\quad\quad\quad ok \leftarrow false; \\
    &\quad\quad\text{end} \\
    &\quad\text{end} \\
  \end{align*}
if \( ok \leftarrow true \) then
  \begin{align*}
    &\text{AddStatements}(D, \text{TempModel}); \\
  \end{align*}
end

Algorithm 9: Performing a content manipulation removal query

\textbf{input} : \( D, Q \)

\textbf{initialization:} \( \text{TempModel} \leftarrow \emptyset, Q_E = \text{PrepareExtentQuery}(Q), ok = true; \)
\( \text{ExtTable} \leftarrow \text{TableQuery}(D, Q_E); \)
if \( \text{IsOK(ExtTable)} \) then
  \begin{align*}
    &\text{forall } s \in \text{ConstructGP}(Q) \text{ do} \\
    &\quad \text{if } ok = true \text{ then} \\
    &\quad\quad \text{Statements} \leftarrow \text{MakeRemovalStatements}(s, \text{ExtTable}); \\
    &\quad\quad \text{if } \text{IsOK(Statements)} \text{ then} \\
    &\quad\quad\quad \text{AddStatements(\text{TempModel}, \text{Statements})}; \\
    &\quad\quad\text{else} \\
    &\quad\quad\quad ok \leftarrow false; \\
    &\quad\quad\text{end} \\
    &\quad\text{end} \\
  \end{align*}
if \( ok \leftarrow true \) then
  \begin{align*}
    &\text{RemoveStatements}(D, \text{TempModel}); \\
  \end{align*}
end
Figure 7.4: UML static class diagram of HPG servlet
Figure 7.5: UML message sequence chart of an example page request
Figure 7.6: DM Builder screenshot

Figure 7.7: AM Builder screenshot
Figure 7.8: Defining articulations in AT Mapper
Chapter 8
Concluding Remarks

This work is aimed at bringing the Hera method, intended from the beginning to support the next generation of (Semantic) Web Information Systems, into a new usability level that would turn the method into a universal and full-fledged design method. In order to do so the following extensions have been introduced: the original AM has been refined to support more flexible data retrieval facilities, an advanced AM has been introduced by extending the basic AM facilities, models supporting the design and reuse of complex WIS have been proposed, a prototype of the Hera HPG engine supporting data content manipulations has been implemented and a set of design support tools have been prototyped. Section 8.1 revisits the research topics stated in Chapter 1 and summarizes how they were handled in this dissertation. Section 8.2 sketches directions of possible future research related to the mentioned research topics.

8.1 Conclusions

In this section we discuss the research topics described in Chapter 1. We summarize how this work contributes to these particular research paths.

**Research Topic 1: Application logic and its specification in the context of WIS**

Chapter 2 explains the term application logic in the context of our interest. The presentation logic capturing the virtue of user interaction with the web interface is a distinguishing factor of WIS. However, the functionality of WIS includes more aspects that are perceived by users. By the term business logic we mean all operations possibly initiated by (not only) user interaction with WIS but not directly visible (data updates, synchronization of processes). The term application logic in this dissertation covers both these aspects — the presentation and business logic. The models expressing WIS application logic are discussed in almost all chapters. They are the domain and application model, and the additional models needed for supporting the automated generation of that application model from specifications, thereby focusing on the application logic from different points of view (including the collaboration model discussed in Chapter 5).
Concluding Remarks

Research Topic 2: Modeling Web interfaces built on top of data domains
Chapters 3 and 4 describe application modeling techniques based on runtime retrieval of data needed for the process of populating Web pages. Although the runtime instantiation of pages depends also on the system architecture that is used (and Hera uses such architecture explained in Chapter 1), the developed specification techniques should also support it. More advanced techniques for the dynamic adaptation of web interfaces are explained in Chapter 4. Navigation unit inheritance allows a universal and elegant means of runtime interface adaptation. Advanced facilities such as inclusion conditions and conditional navigation techniques provide designers with countless possibilities for the specification of this adaptation.

Research Topic 3: Full-fledged WIS application modeling
Modern WIS require not only an advanced Web front-end (Web interface exposed to users) but also an advanced back-end — facilities allowing processing of the user’s feedback and storing the possible result in a (persistent) data repository. Chapter 4 extends the Hera AM with an ECA rule framework that allows the separation of the navigation structure specification from the business logic specification. Rules can update the application data content and allow communication with external systems. Entry forms introduced into AM navigation models improve the user interaction with WIS.

Research Topic 4: Modeling of WIS requiring extensive user collaboration
Possible problems with modeling WIS that must synchronize interaction with different types of users or systems, using navigation driven AM, are stressed in Chapter 5. For this purpose a collaboration model (an extended UML activity diagram) is adopted into Hera and extended with facilities allowing full functional specification at the level of WM. The automated procedure for generating AM from WM described in Chapter 5 helps to minimize the design and maintenance costs.

Research Topic 5: Reuse of application models in the context of the Hera method
Saving design effort and costs by reusing software specification artifacts has an important place in software development. Reusing AM specifications for multiple domains in a template-like fashion is discussed in Chapter 6. So called Application Templates allow to model generic parts of Web applications that can be semi-automatically deployed in different domains by using mappings from template concepts to concrete domain concepts as template parameters.

Research Topic 6: Validation of proposed models
Software tools used in Hera are summarized in Chapter 7. The list of existing software tools begins with the Hera Presentation Generator that is the Hera runtime engine. This engine is a Web server that uses Hera models (DM, AM) for the online generation of pages based on user requests. Design support is provided by visual DM and AM Builders and AT mapping is facilitated by using the AT Mapper.
8.2 Future Work

Research is an ongoing process: every new step opens a plethora of possible new ways to explore. This is even more true in the field of software engineering and methods where the research itself is accompanied by the development of suitable supporting tools or tools proving the validity of proposed concepts. The following directions of future research — and certainly many more — can be further developed:

- Using Semantic Web models in all Hera models opens wide possibilities of model verification and validation that can be performed at design time. In this way possible defects coming into light in full operation of WIS can be detected in early stages of the design. This is very useful especially for an AM that is critical for correct functionality. Extended models such as OWL [McGuiness and van Harmelen, 2004] that are based on RDFS introduce even more special properties (e.g. cardinalities of properties, functional properties, etc.) that can be used for model verification.

- The content of the Web is dynamic and so are constantly evolving services offered by the Web. One of commonly accepted concepts of the Semantic Web future is a vision of the Web where WIS dynamically and intelligently locate the Web Services (WS) they need. Another challenging research direction is using loosely coupled WS for WIS specification such that, instead of a concrete location of the service and exact parameter specification, only requirements for a service would be specified and the WIS would find and explore such services all by itself.

- The security and better organization of the Web is highly required by many service providers but current methods pay only little (if any) attention to this issue (including Hera). This issue is very serious because it goes so deep as to involve insufficiencies of the basic protocols used to run the Web. Design methods and tools however should be also aware of this phenomenon and be prepared to cope with it. The question of trusted internet transactions is even more important in the context of intelligent and dynamic service WS exploration.

- Aspect-orientation is a new modeling paradigm and a new trend in software modeling that allows a high level of concern separation in a single software model. This may prove extremely useful for WIS modeling due to its nature and due to severe and numerous requirements on WIS models. While different models specify essentially different facets of the designed WIS, different aspects (present in all models) may form an orthogonal separation of cross-cutting concerns applied to already existing models (for example the personalization aspect that may apply to all models).
Bibliography


List of Figures

2.1 The architecture of Hera WIS ............................................. 15
2.2 The Hera design support .................................................. 16

3.1 The structure of a data domain for the conference submission and review system .................................................. 18
3.2 An example of a graphical representation of a domain ............... 19
3.3 An example of a vocabulary ................................................. 20
3.4 A graphical representation of a simple navigation view over the conference submission domain ............................................. 38

4.1 A general structure of AM .................................................. 40
4.2 The basic navigation meta-model expressed as RDFS vocabulary ... 41
4.3 The example of AM graphical notation (left) and its RDFS representation .................................................. 42
4.4 The extension of navigation meta-model accommodating input forms ... 44
4.5 An example of a form and an inclusion condition presented in the Hera AM notation (left), and as RDFS graph (right) ....................... 46
4.6 An example of conditional navigation presented using the Hera AM notation (left), and RDFS (right) ............................................. 49
4.7 An example of the NU specialization ........................................ 50
4.8 Representation of a “follow link” event declaration using abstract syntax (left) and AM vocabulary (right). ................................. 53
4.9 Representation of a submission event declaration using the abstract syntax (left) and AM vocabulary (right). ................................. 54
4.10 Representation of a timed event declaration using abstract syntax (left) and AM vocabulary (right) ............................................. 55
4.11 Representation of an exception event declaration using abstract syntax (left) and AM vocabulary (right). ................................. 55
4.12 Representation of an external event declaration using the abstract syntax (left) and the AM vocabulary (right). ................................. 56
4.13 Representation of an exception event declaration using abstract syntax (left) and AM vocabulary (right). ................................. 57
4.14 Representation of a rule with a data manipulation query using the abstract syntax (left) and AM vocabulary (right). ................................. 60
4.15 Representation of an external WS call providing ISBN information represented using the abstract syntax (left) and AM vocabulary (right) .......................................................... 62
4.16 Representation of timer initialization using the abstract syntax (left) and AM vocabulary (right) .......................................................... 63
4.17 The simplified content manipulation meta-model ........................................... 63
4.18 An example of AM using Hera AM notation .................................................. 66
4.19 RDFS representation of the Login NU ............................................................ 66
4.20 RDFS representation of the Review ForReviewer NU ....................................... 69

5.1 Process-driven design of WIS ..................................................................... 74
5.2 Task model visualized using the use case diagram ......................................... 75
5.3 Example CS ....................................................................................... 79
5.4 Triggering parallel execution branches ......................................................... 83
5.5 Joining parallel execution branches .............................................................. 83
5.6 Triggering optional execution branches ......................................................... 84
5.7 Merging alternative execution branches ......................................................... 84
5.8 Generation of AM from CM .................................................................... 91
5.9 Context data model ............................................................................. 93
5.10 Message queue example ....................................................................... 94
5.11 Sequence of activities for the MakeSubmission task of Author ............... 102
5.12 An example of the navigation structure generated from a workflow ........ 103
5.13 Example of generated NU ..................................................................... 106

6.1 An AT linked with the destination AM ......................................................... 112
6.2 Deployment of a single application template to two different applications (and domains) using two different mappings ........................................ 113
6.3 Publication TDM (a) and the target DM (b) .................................................. 117
6.4 The navigation model of the publication TAM ............................................. 118
6.5 Deployed Publications NT .................................................................... 120
6.6 Guided Tour TDM (top) and navigation model (bottom) .............................. 123
6.7 Guided Tour transformed for the deployment within the conference system AM ....................................................................................... 126
6.8 Guided Tour deployed in the conference system AM ................................... 127
6.9 RDFS vocabulary for mappings .................................................................. 128

7.1 HPG initial demo page (a) and login page for the publications demo (b) .... 133
7.2 Activity diagram showing the collaboration of users and web applications using HPG .................................................................................. 134
7.3 Retrieval table query (left) and retrieval graph query (right) ...................... 139
7.4 UML static class diagram of HPG servlet .................................................. 152
7.5 UML message sequence chart of an example page request ....................... 153
7.6 DM Builder screenshot .......................................................................... 154
7.7 AM Builder screenshot .......................................................................... 154
7.8 Defining articulations in AT Mapper ................................................. 155
Index

Abstract Syntax, 23
Activity, 77
Activity Clusters, 101
Activity Diagrams, 72
Activity State, 77
AM Generation Procedure, 105
Application Model, 13
  Advanced AM, 39
    Content Manipulation Model, 43
    Navigation Model, 43
  Basic AM, 17
Application Template, 109
Articulation, 126
Attribute, 26

Binding Variable, 24
BPML, 72
Business Process, 71

Collaboration Model, 76
Composed Events, 82
Composition, 26
  Data Composition, 28
Conditional Navigation, 47

Decision Block, 81
Domain, 17
  Domain Model, 21

ECA Rules, 41
Entity-Relationship Modeling, 18
Event Order, 103
  Event Order Operator, 103
Events, 40

Fork Block, 81
Form, 43

Graph Pattern, 24
  Data Path, 25
  Path Expression, 24
Guarding Condition, 51

Hera, 2
  Hera Architecture, 14
    Application Layer, 14
    Presentation Layer, 14
    Semantic Layer, 14
Inclusion Condition, 45
Join Block, 81

Link, 26
  Data Link, 30
LTS, 76

Merge Blocks, 84
Message Queues, 89
Messages, 89

Navigation Clusters, 101
NU - Navigation Unit, 26
NU Relationship, 26

OO-H, 11
OOHDM, 9

Parameterized Query, 27
Petri Nets, 72
Process Algebra, 72
  CCS, 72
  CSP, 72

Retrieval Query, 25
RMM, 7
INDEX

TAM, 110
Task Model, 75
Task Partitioning, 103
TDM, 110

UML, 8
UWE, 10

Vocabulary, 19

Wait State, 78
Web Information Systems, 1
WebML, 8
WSDM, 12
Summary

The importance of the World Wide Web has grown tremendously over the past decade (or decade and a half). With a quickly growing amount of information published on the Web and its rapidly growing audience, requirements put on Web-based Information Systems (WIS), their developers and maintainers have grown significantly. Nowadays it is not enough to only provide fresh and structured information but in order to be competitive it is necessary to provide users with complete (and often complex) information services. This rapid movement caused a gap between the huge demand for services hosted by the Web and the availability of suitable and standardized Web development methods. Software practices have shown that design methods for traditional information systems alone are not sufficient for modeling WIS. The most obvious reason is the unique nature of the Web interface that asks for an explicit model. Nevertheless, a perhaps even more important distinction is a huge and diverse audience of WIS that is often unknown at design time. Hence, the need of a suitable generic design method for WIS is dire.

A number of methods aimed at WIS design and development have been proposed and most of them rely on proven concepts from known traditional software design methods. The Hera method is one of few methods proposed originally for WIS software design. Moreover, it is one of the first methods of which all models are expressed in RDFS, an essential modeling language of the Semantic Web. The application model used in the Hera method that originally expressed only the WIS navigation structure has evolved into a fully-fledged application model defining the complete functionality of WIS where the front-end navigation specification is separated from the data content manipulation specification at the back-end. The subject of this dissertation is this extension of the Hera method together with a proposal of models supporting the design of complex WIS.

The summary of existing methods presented in Chapter 2 maps the trends in the field of WIS design and focuses on the WIS application logic models (covering the specification of the navigation structure and the specification of data transactions hidden from end users) used in different methods. This summary helps to justify the discussed extension of the Hera framework, the Hera application model (AM), and the additional design support models. The main concepts of domain and application modeling used in Hera are discussed in Chapter 3. The vocabulary of application model meta-concepts introduced there forms the so called basic AM that provides a well structured specification means for the Web interface of data content described by the Domain Model (DM). The advanced AM intro-
duced in Chapter 4 extends the basic AM with a Content Manipulation Model (CMM). For the specification of data content manipulations on RDF data domains we have extended the SeRQL RDF/RDFS query language with data manipulation constructs. The navigation specification expressed by the basic AM is called the Navigation Model (NM). It is enriched with the notion of forms allowing users to enter information. Inclusion conditions can be used in the NM as well to allow runtime adaptation of the navigation structure.

Models to support application modeling are discussed in Chapters 5 and 6. Many WIS require collaboration of multiple users in order to satisfy their purpose. A good example is a conference submission and review system where users, playing different roles, must exchange different types of messages (e.g. papers and reviews) in an asynchronous way regardless of the duration of single browsing sessions. For these systems a collaboration model explicitly specifying the process of collaboration is introduced in Chapter 5. This model is used for the automated generation of AM. Another type of design support is presented in Chapter 6. In WIS (and not only there) one can observe often occurring functional patterns. That naturally leads to the idea of reusing functional specification for different applications. The approach presented in this dissertation allows the complete reuse and repeated deployment of (parts of) an AM for different domains in a template manner. So called Application Templates (AT) are AM that are specified using Template Domain Models (TDM), virtual DM that mimic parts of assumed domains. Chapter 6 describes the deployment of AT for concrete domains by using template parameters that in this case are mappings of TDM concepts to concrete domain concepts. This process can be automated.

An important part of the research in the field of design methods is the validation of proposed modeling concepts through software prototypes allowing to use proposed methods and to evaluate them. Software prototypes and concepts summarized in Chapter 7 give us a valuable feedback on the direction of this research. The most important is the Hera Presentation Generator (HPG), a Web server that deploys Hera models for the runtime generation of Web pages. HPG has been implemented as a Java servlet and it runs under the Apache Tomcat Web server. It uses the RDF data repository Sesame and (among other tasks) it performs SeRQL-like content manipulation queries. The support for designers is provided by visual model builders (Visio based DM/AM builders and a Java based AT Mapper).
Samenvatting

Doorheen de laatste tien tot vijftien jaar kende het Web een ongelofelijk groei. Met een snel groeiende hoeveelheid informatie gepubliceerd op het Web in combinatie met een sterk groeiende publiek, heeft de druk op Web-gebaseerde Informatie Systemen (WIS), de ontwikkelaars en de systeembeheerders sterk doen toenemen. Tegenwoordig volstaat het niet meer alleen om dynamische en gestructureerde informatie weer te geven. Om sterk te staan in de markt is het noodzakelijk dat WIS gebruikers voorzien van complete en soms complexe informatie diensten. Deze snelle beweging heeft een brede kloof tussen enerzijds de grote vraag naar diensten op het Web en anderzijds de beschikbaarheid van geschikte en gestandaardiseerde Web ontwikkelingsmethodes veroorzaakt. Implementatie ervaringen hebben aangetoond dat huidige methodes voor traditionele informatiesystemen alleen, ontoereikend zijn voor het modelleren van WIS. De meest voor de hand liggende rede is de unieke eigenschap van de Web interface die vraagt voor een expliciete modellering. Echter, er is nog een belangrijker onderscheid, tijdens het ontwerp van een WIS is het immense en diverse publiek nog steeds onbekend. Het is duidelijk dat de nood aan een geschikte generische ontwerpmethode voor WIS groot is.

Sommige technieken, die gebaseerd zijn op bewezen concepten van bestaande traditionele software design methodes, worden nu ook voorgesteld als mogelijkheid voor WIS design en ontwikkeling. De Hera methode is een van de enige originele methodes ontwikkeld voor WIS software design. Meer zelfs, het is een van de eerste methodes waarin alle modellen zijn uitgedrukt in RDFS, een essentiële modelleringstaal van het Semantic Web. Het applicatie model, dat gebruikt word in de Hera methode en origineel alleen de WIS navigatie structuur uitdrukte, is geëvolueerd tot een volledige model die de complete WIS functionaliteit definieert waarbij de front-end navigatie specificatie gescheiden is van de content manipulatie specificatie aan de back-end. Het onderwerp van deze dissertatie is de uitbreiding van de Hera methode tezamen met een modelleringsvoorstel ter ondersteuning van het ontwerp van complexe WIS.

De samenvatting van bestaande methodes beschreven in Hoofdstuk 2 relateert de trends in WIS design en bekijkt de WIS applicatie logica modellen (gebruikt om de specificatie van de navigatie structuur en de specificatie van de data transities, verborgen voor de eind gebruikers, uit te drukken) gebruikt in verschillende methodes. Deze samenvatting helpt om de voorgestelde uitbreiding van het Hera framework, het Hera applicatie model (AM) en de additionele design ondersteuningsmodellen te verantwoorden. De hoofdconcepten
van de domein- en applicatiemodellering gebruikt in Hera worden uitgelegd in Hoofdstuk 3. De vocabulaire van AM meta-concepten hier geïntroduceerd vormt het zogenaamde basis AM dat een goed gestructureerde specificatie voor de Web interface van data content, beschreven door het Domein Model (DM), ter beschikking stelt. Het geavanceerde AM geïntroduceerd in Hoofdstuk 4 breidt het basis AM uit met een Content Manipulatie Model (CMM). Voor de specificatie van deze data content manipulaties op RDF data domeinen hebben we de RDF/RDFS query taal — SeRQL — uitgebreid met data manipulatie constructies. De navigatie specificatie uitgedrukt in het basis AM noemen we het Navigatie Model (NM). Het is uitgebreid met elektronische formulieren die de gebruiker de mogelijkheid geven om informatie in te vullen. Inclusie voorwaarden kunnen in het NM ook gebruikt woorden om adaptatie van de navigatie structuur toe te laten tijdens runtime.

Modellen ter ondersteuning van de applicatie modellering worden besproken in Hoofdstuk 5 en 6. Om hun doel te vervullen zijn verschillende WIS afhankelijk van de collaboratie tussen verschillende gebruikers. Een goed voorbeeld hiervan is een conferentie artikel review systeem waarin gebruikers, in verschillende rollen, verschillende boodschappen moeten uitwisselen (bijvoorbeeld artikels en reviews) op een asynchrone manier onafhankelijk van de duur van één sessie. Voor dit type van systemen is een collaboratie model dat expliciet het collaboratie proces specificereert geïntroduceerd in Hoofdstuk 5. Dit model is gebruikt voor de automatische generatie van een AM. Een ander type van design ondersteuning word bediscussieerd in Hoofdstuk 6. In WIS (en niet alleen daar) kan men regelmatig voorkomende functionele patronen observeren. Dit leidt van nature naar het idee om de functionele specificatie voor verschillende applicaties te hergebruiken. De techniek gepresenteerd in deze dissertatie laat het complete hergebruik en het herhaalde lanceren van (delen van) een AM voor verschillende domeinen toe door middel van sjablonen. De zogenaamde applicatie sjablonen (Application Templates, AT) zijn AM dewelke gespecifieerd zijn door middel van sjabloon domein modellen (Template Domain Models, TDM), virtuele DM die delen van aangenomen domeinen nabootsen. Hoofdstuk 6 beschrijft de lancering van AT voor concrete domeinen door gebruik te maken van sjabloonparameters die in dit geval concepten van het TDM relateren naar concrete domein concepten. Dit proces kan geautomatiseerd worden.

Een belangrijk onderdeel in het onderzoeksgebied van ontwerp methodes is de validatie van de voorgestelde modelleringsconcepten door middel van software prototypes die het toelaten om de voorgestelde methodes te gebruiken en te evalueren. De software prototypes en concepten samengevat in Hoofdstuk 7 geven ons waardevolle feedback op de richting van dit onderzoek. Het centrale prototype is de Hera Presentatie Generator (HPG), een Web server die Hera modellen gebruikt voor de runtime generatie van Web pagina’s. HPG is geïmplementeerd als een Java servlet en draait onder de Apache Tomcat Web server. Het gebruikt de RDF database Sesame en voert onder andere taken zoals, op SeRQL gelijkende, content manipulatie queries uit. De ondersteuning voor ontwerpers wordt aangeboden door middel van visuele model constructie software (onder andere een Visio gebaseerd DM/AM constructie pakket en een Java gebaseerde AT Mapper).
Curriculum Vitae

Peter Barna was born on 13 September 1971 in Krompachy, Slovakia. After completing comprehensive high school (Gymnázium kpt. Jána Nálepk) he started his study at the Technical University of Košice at the Department of Electrical Engineering and Computer Science. He graduated in 1994 and received the master degree in Electrical Engineering and Computer Science.

Between the years 1994 and 1998 he worked as a software engineer in a startup company designing and deploying geographical information systems for municipal authorities. In order to acquire more knowledge and skills in managing software projects he joined the software technology post-master program at the Stan Ackermans Institute at the Eindhoven University of Technology. After one year of study and one year of industrial project in a California based company developing text search solutions for Intranet and Internet he received the degree of Professional Doctorate in Engineering (PDEng). In 2002 he started his PhD research at Eindhoven University of Technology in the group of Databases and Hypermedia. In 2006 he started to work as software engineer in Topic Automatisering. His current project in Philips Research aims at the development of next-generation user interface devices.
1998

1998-1  Johan van den Akker (CWI)  
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2  Floris Wiesman (UM)  
Information Retrieval by Graphically Browsing Meta-Information

1998-3  Ans Steuten (TUD)  
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4  Dennis Breuker (UM)  
Memory versus Search in Games

1998-5  E.W.Oskamp (RUL)  
Computerondersteuning bij Straftoemeting

1999

1999-1  Mark Sloof (VU)  
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2  Rob Potharst (EUR)  
Classification using decision trees and neural nets

1999-3  Don Beal (UM)  
The Nature of Minimax Search

1999-4  Jacques Penders (UM)  
The practical Art of Moving Physical Objects

1999-5  Aldo de Moor (KUB)  
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems
1999-6 Niek J.E. Wijngaards (VU)  
Re-design of compositional systems
1999-7 David Spelt (UT)  
Verification support for object database design
1999-8 Jacques H.J. Lenting (UM)  
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

====

2000
====

2000-1 Frank Niessink (VU)  
Perspectives on Improving Software Maintenance
2000-2 Koen Holtman (TUE)  
Prototyping of CMS Storage Management
2000-3 Carolien M.T. Metselaar (UVA)  
Sociaal-organisatorische gevolgen van kennis technologie; een procesbenadering en actorperspectief.
2000-4 Geert de Haan (VU)  
ETAG, A Formal Model of Competence Knowledge for User Interface Design
2000-5 Ruud van der Pol (UM)  
Knowledge-based Query Formulation in Information Retrieval
2000-6 Rogier van Eijk (UU)  
Programming Languages for Agent Communication
2000-7 Niels Peek (UU)  
Decision-theoretic Planning of Clinical Patient Management
2000-8 Veerle Coup (EUR)  
Sensitivity Analysis of Decision-Theoretic Networks
2000-9 Florian Waas (CWI)  
Principles of Probabilistic Query Optimization
2000-10 Niels Nes (CWI)  
Image Database Management System Design Considerations, Algorithms and Architecture
2000-11 Jonas Karlsson (CWI)  
Scalable Distributed Data Structures for Database Management
<table>
<thead>
<tr>
<th>Year</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Qualitative Approaches to Quantifying Probabilistic Networks</td>
<td>Silja Renooij (UU)</td>
</tr>
<tr>
<td></td>
<td>Agent Programming Languages: Programming with Mental Models</td>
<td>Koen Hindriks (UU)</td>
</tr>
<tr>
<td></td>
<td>Learning as problem solving</td>
<td>Maarten van Someren (UvA)</td>
</tr>
<tr>
<td></td>
<td>Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets</td>
<td>Evgueni Smirnov (UM)</td>
</tr>
<tr>
<td></td>
<td>Processing Structured Hypermedia: A Matter of Style</td>
<td>Jacco van Ossenbruggen (VU)</td>
</tr>
<tr>
<td></td>
<td>Task-based User Interface Design</td>
<td>Martijn van Welie (VU)</td>
</tr>
<tr>
<td></td>
<td>Diva: Architectural Perspectives on Information Visualization</td>
<td>Bastiaan Schonhage (VU)</td>
</tr>
<tr>
<td></td>
<td>A Compositional Semantic Structure for Multi-Agent Systems Dynamics</td>
<td>Pascal van Eck (VU)</td>
</tr>
<tr>
<td></td>
<td>Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes</td>
<td>Pieter Jan 't Hoen (RUL)</td>
</tr>
<tr>
<td></td>
<td>Modeling and Simulating Work Practice</td>
<td>Maarten Sierhuis (UvA)</td>
</tr>
<tr>
<td></td>
<td>BRAHMS: a multiagent modeling and simulation language for work practice analysis and design</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Knowledge Management: The Role of Mental Models in Business Systems Design</td>
<td>Tom M. van Engers (VUA)</td>
</tr>
<tr>
<td>2002</td>
<td>Architecture-Level Modifiability Analysis</td>
<td>Nico Lassing (VU)</td>
</tr>
<tr>
<td></td>
<td>Modelling and searching web-based document collections</td>
<td>Roelof van Zwol (UT)</td>
</tr>
<tr>
<td></td>
<td>Database Optimization Aspects for Information Retrieval</td>
<td>Henk Ernst Blok (UT)</td>
</tr>
</tbody>
</table>
2002-04 Juan Roberto Castelo Valdueza (UU)
   The Discrete Acyclic Digraph Markov Model in Data Mining
2002-05 Radu Serban (VU)
   The Private Cyberspace Modeling Electronic
   Environments inhabited by Privacy-concerned Agents
2002-06 Laurens Mommers (UL)
   Applied legal epistemology; Building a knowledge-based ontology of
   the legal domain
2002-07 Peter Boncz (CWI)
   Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications
2002-08 Jaap Gordijn (VU)
   Value Based Requirements Engineering: Exploring Innovative
   E-Commerce Ideas
2002-09 Willem-Jan van den Heuvel (KUB)
   Integrating Modern Business Applications with Objectified Legacy
   Systems
2002-10 Brian Sheppard (UM)
   Towards Perfect Play of Scrabble
2002-11 Wouter C.A. Wijngaards (VU)
   Agent Based Modelling of Dynamics: Biological and Organisational
   Applications
2002-12 Albrecht Schmidt (Uva)
   Processing XML in Database Systems
2002-13 Hongjing Wu (TUE)
   A Reference Architecture for Adaptive Hypermedia Applications
2002-14 Wieke de Vries (UU)
   Agent Interaction: Abstract Approaches to Modelling, Programming
   and Verifying Multi-Agent Systems
2002-15 Rik Eshuis (UT)
   Semantics and Verification of UML Activity Diagrams for
   Workflow Modelling
2002-16 Pieter van Langen (VU)
   The Anatomy of Design: Foundations, Models and Applications
2002-17 Stefan Manegold (UVA)
   Understanding, Modeling, and Improving Main-Memory Database
   Performance

===
2003
===

2003-01 Heiner Stuckenschmidt (VU)
   Ontology-Based Information Sharing in Weakly Structured Environments
2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions
2004-01 Virginia Dignum (UU)
   A Model for Organizational Interaction: Based on Agents, 
   Founded in Logics
2004-02 Lai Xu (UvT)
   Monitoring Multi-party Contracts for E-business
2004-03 Perry Groot (VU)
   A Theoretical and Empirical Analysis of Approximation in Symbolic 
   Problem Solving
2004-04 Chris van Aart (UVA)
   Organizational Principles for Multi-Agent Architectures
2004-05 Viara Popova (EUR)
   Knowledge discovery and monotonicity
2004-06 Bart-Jan Hommes (TUD)
   The Evaluation of Business Process Modeling Techniques
2004-07 Elise Boltjse (UM)
   Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar 
   abstract denken, vooral voor meisjes
2004-08 Joop Verbeek(UM)
   Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale 
   politie gegevensuitwisseling en digitale expertise
2004-09 Martin Caminada (VU)
   For the Sake of the Argument; explorations into argument-based 
   reasonings
2004-10 Suzanne Kabel (UVA)
   Knowledge-rich indexing of learning-objects
2004-11 Michel Klein (VU)
   Change Management for Distributed Ontologies
2004-12 The Duy Bui (UT)
   Creating emotions and facial expressions for embodied agents
2004-13 Wojciech Jamroga (UT)
   Using Multiple Models of Reality: On Agents who Know how to Play 
   Logic in Conflict. Logical Explorations in Strategic Equilibrium
2004-14 Paul Harrenstein (UU)
   Multi-Relational Data Mining
2004-15 Federico Divina (VU)
   Hybrid Genetic Relational Search for Inductive Learning
2004-16 Mark Winands (UM)
   Informed Search in Complex Games
2004-18 Vania Bessa Machado (UvA)
   Supporting the Construction of Qualitative Knowledge Models
2004-19 Thijs Westerveld (UT)
   Using generative probabilistic models for multimedia retrieval
2004-20 Madelon Evers (Nyenrode)
   Learning from Design: facilitating multidisciplinary design teams

====

2005
====

2005-01 Floor Verdenius (UVA)
   Methodological Aspects of Designing Induction-Based Applications
2005-02 Erik van der Werf (UM))
   AI techniques for the game of Go
2005-03 Franc Grootjen (RUN)
   A Pragmatic Approach to the Conceptualisation of Language
2005-04 Nirvana Meratnia (UT)
   Towards Database Support for Moving Object data
2005-05 Gabriel Infante-Lopez (UVA)
   Two-Level Probabilistic Grammars for Natural Language Parsing
2005-06 Pieter Spronck (UM)
   Adaptive Game AI
2005-07 Flavius Frasincar (TUE)
   Hypermedia Presentation Generation for Semantic Web Information Systems
2005-08 Richard Vdovjak (TUE)
   A Model-driven Approach for Building Distributed Ontology-based Web Applications
2005-09 Jeen Broekstra (VU)
   Storage, Querying and Inferencing for Semantic Web Languages
2005-10 Anders Bouwer (UVA)
   Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments
2005-11 Elth Ogston (VU)
   Agent Based Matchmaking and Clustering - A Decentralized Approach to Search
2005-12 Csaba Boer (EUR)
   Distributed Simulation in Industry
2005-13 Fred Hamburg (UL)
   Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen
2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

2006

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)
XML schema matching -- balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web
2006-09 Mohamed Wahdan (UM)
  Automatic Formulation of the Auditor's Opinion
2006-10 Ronny Siebes (VU)
  Semantic Routing in Peer-to-Peer Systems
2006-11 Joeri van Ruth (UT)
  Flattening Queries over Nested Data Types
2006-12 Bert Borgers (VU)
  Interactivation - Towards an e-cology of people, our technological
  environment, and the arts
2006-13 Henk-Jan Lebbink (UU)
  Dialogue and Decision Games for Information Exchanging Agents
2006-14 Johan Hoorn (VU)
  Software Requirements: Update, Upgrade, Redesign - towards
  a Theory of Requirements Change
2006-15 Rainer Malik (UU)
  CONAN: Text Mining in the Biomedical Domain
2006-16 Carsten Riggelsen (UU)
  Approximation Methods for Efficient Learning of Bayesian Networks
2006-17 Stacey Nagata (UU)
  User Assistance for Multitasking with Interruptions on a Mobile Device
2006-18 Valentin Zhizhkun (UVA)
  Graph transformation for Natural Language Processing
2006-19 Birna van Riemsdijk (UU)
  Cognitive Agent Programming: A Semantic Approach
2006-20 Marina Velikova (UvT)
  Monotone models for prediction in data mining
2006-21 Bas van Gils (RUN)
  Aptness on the Web
2006-22 Paul de Vrieze (RUN)
  Fundaments of Adaptive Personalisation
2006-23 Ion Juvina (UU)
  Development of Cognitive Model for Navigating on the Web
2006-24 Laura Hollink (VU)
  Semantic Annotation for Retrieval of Visual Resources
2006-25 Madalina Drugan (UU)
  Conditional log-likelihood MDL and Evolutionary MCMC
2006-26 Vojkan Mihajlovic (UT)
  Score Region Algebra: A Flexible Framework for Structured Information
  Retrieval
2006-27 Stefano Bocconi (CWI)
  Vox Populi: generating video documentaries from semantically
  annotated media repositories
2006-28 Borkur Sigurbjörnsson (UVA)
Focused Information Access using XML Element Retrieval

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)
Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance

2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs

2007-07 Natasa Jovanovic’ (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings

2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations

2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation

2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System

2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty

2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology

2007-14 Niek Bergboer (UM)
Context-Based Image Analysis
2007-15 Joyca Lacroix (UM)
   NIM: a Situated Computational Memory Model
2007-16 Davide Grossi (UU)
   Designing Invisible Handcuffs. Formal investigations in
   Institutions and Organizations for Multi-agent Systems
2007-17 Theodore Charitos (UU)
   Reasoning with Dynamic Networks in Practice
2007-18 Bart Orriens (UvT)
   On the development an management of adaptive business
   collaborations
2007-19 David Levy (UM)
   Intimate relationships with artificial partners
2007-20 Slinger Jansen (UU)
   Customer Configuration Updating in a Software Supply Network
2007-21 Karianne Vermaas (UU)
   Fast diffusion and broadening use: A research on residential
   adoption and usage of broadband internet in the Netherlands
   between 2001 and 2005
2007-22 Zlatko Zlatev (UT)
   Goal-oriented design of value and process models from patterns