An Engineering Method for Adaptive, Context-aware Web Applications

Von der Fakultät für Ingenieurwissenschaften der

Universität Duisburg–Essen

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

genehmigte Dissertation

von

Joachim Wolfgang Kaltz

aus Mainz

Referent: Prof. Dr.-Ing. Jürgen Ziegler

Korreferent: Prof. Dr. Heinz Ulrich Hoppe

Tag der mündlichen Prüfung: 18. Juli 2006

Acknowledgments

I thank the University of Duisburg-Essen and all its employees for providing the framework enabling me to complete this dissertation. Foremost, I wish to thank Professor Jürgen Ziegler for his supervision and regular suggestions for improving the quality of this work.

I am also grateful to my colleagues at the Institute for Interactive Systems and Interaction Design for the pleasant work atmosphere. Particularly valuable were the efforts by Steffen Lohmann and Eike Lang for constructive discussions about the subject matter and for reading corresponding texts with great attention. Last but not least, I thank my parents, Barbara and Norbert Kaltz, for their continuing support over the years.

Finally, I would like to indulge myself by recalling an (obscure) citation of film actor Jean-Claude Van Damme:

> ”L’être humain a tellement l’habitude de parler et de regarder, temps en temps les gens qui sont awares que les gens qui voient, parce qu’ils sont obligés de sentir les sentations, les gens qu’ils aiment pas, qu’ils aiment bien, et les objets qui sont, et on est awares.”

This gem in terms of semantic ambiguity serves to remind me that the human being is presumably the most elaborate existing context-aware piece of machinery, and will perhaps never be surpassed by software in that regard. This author would like to thank all humans assisting in raising our level of context-awareness, be it by works of art or science.

Abstract

Users of Web-based software encounter growing complexity of the software resulting from the increasing amount of information and service offering. As a consequence, the likelihood that users employ the software in a manner compatible with the provider's interest decreases. Depending on the purpose of the Web application, a provider's goal can be to guide and influence user choices in information and service selection, or to assure user productivity. An approach at addressing these goals is to adapt the software's behavior during operation to the context in which it is being used. The term context-awareness originates in mobile computing, where research projects have studied context recognition and adaptation in specific scenarios. Context-awareness is now being studied in a variety of systems, including Web applications. However, how to account for context in a Web Engineering process is not yet established, nor is a generic means of using context in a Web software architecture.

This dissertation addresses the question of how context-awareness can be applied in a general-purpose, systematic process for Web application development: that is, in a Web Engineering process. A model for representing an application's context factors in ontologies is presented. A general-purpose methodology for Web Engineering is extended to account for context, by putting in relation context ontologies with elements of the application domain. The application model is extended with adaptation specifications, defining at which places in the application adaptation to context is to occur, and according to what strategy.

Application and context models are system interpretable, in order to support automatic adaptation of a system's behavior during its operation, that is, consequently to user requests. Requirements for a corresponding Web software architecture for context are established first at the conceptual level, then specifically in a content-based architecture based on an XML stack. The CATWALK software framework, an implementation of an architecture enabling adaptation to context is described. The framework provides mechanisms for interpreting application and context models to generate an adaptive application, meaning to generate responses to user requests, where the generation process makes decisions based on context information. For this purpose, the framework contains default implementations for context recognition and adaptation mechanisms.

The approach presented supports a model-based development of Web applications which adapt to context. The CATWALK framework is an implementation for model interpretation in a run-time system and thus simplifies the development of Web applications which adapt to context. As the framework is component-based and follows a strict separation of concerns, the default mechanisms can be extended or replaced, allowing to reduce the amount of custom code required to implement specific context-aware Web applications or to study alternative context inference or adaptation strategies. The use of the framework is illustrated in a case study, in which models are defined for a prototypical application, and this application is generated by the framework. The purpose of the case study is to illustrate effects of adaptation to context, based on context description and adaptation specifications in the application model.

Contents

1 Introduction **13**
- 1.1 Theme Explanation . . . 13
  - 1.1.1 Adaptivity and Context . . . 14
  - 1.1.2 Web Applications . . . 15
  - 1.1.3 Web Engineering . . . 16
- 1.2 Problems and Challenges . . . 19
- 1.3 Goals of this Dissertation . . . 20

2 State of the Art **23**
- 2.1 Context-awareness . . . 23
  - 2.1.1 Accounting for Context . . . 24
  - 2.1.2 Context Representation . . . 27
  - 2.1.3 Architectures for Adaptivity . . . 28
- 2.2 Common Approaches to Web Application Development . . . 30
  - 2.2.1 Portal Technology . . . 31
  - 2.2.2 Web Content Management Systems . . . 31
  - 2.2.3 Web Services . . . 32
  - 2.2.4 Application Frameworks . . . 35
- 2.3 Methodologies for Web Engineering . . . 39
- 2.4 Engineering for Context-awareness and Usability . . . 42

3 Context in Web Applications **45**
- 3.1 Adaptation to Context . . . 45
- 3.2 Context Usages and Characteristics . . . 46
  - 3.2.1 Example Scenarios . . . 47
  - 3.2.2 Categorization of Context Elements . . . 49
  - 3.2.3 Aspects of Context . . . 51
- 3.3 A Unifying Context Model . . . 53
  - 3.3.1 A Formal Definition of Context . . . 53
  - 3.3.2 Concrete Context Model . . . 55
  - 3.3.3 Context Knowledge and Relations . . . 56
  - 3.3.4 Example Context Models . . . 57
- 3.4 Context Modeling in the Web Engineering Process . . . 59
- 3.5 Application Models and Context . . . 62
  - 3.5.1 Domain Ontology . . . 62
  - 3.5.2 Context Representation . . . 63

3.5.3 Ontology Lookups . . . 64
3.5.4 Adaptation Specifications . . . 65
3.5.5 Navigation Model . . . 66
3.5.6 Modeling for Presentation . . . 67
3.6 Model Visualization and Editing . . . 68
3.7 Summary . . . 69

4 A Web Software Architecture for Context 71
4.1 Requirements for an Adaptive Run-time System . . . 71
4.1.1 System Environment . . . 71
4.1.2 Reference Architecture . . . 73
4.1.3 Web Application Elements . . . 74
4.2 Processing and Architectural Approaches . . . 75
4.2.1 Information Processing . . . 75
4.2.2 Architectures for Adaptive Systems . . . 76
4.2.3 Component-based Software Engineering and Adaptation . . . 77
4.2.4 Processing in the Cocoon Framework . . . 78
4.2.5 Using Cocoon for Adaptivity . . . 80
4.2.6 Components in Cocoon . . . 81
4.3 Principles of the Web Software Architecture for Context . . . 83
4.3.1 Processing Model . . . 84
4.3.2 Architectural Components . . . 86
4.3.3 Domain-specific Components . . . 87
4.3.4 Navigation and Patterns for Presentation . . . 90
4.4 Summary . . . 93

5 The CATWALK Context Framework 95
5.1 Architectural Overview . . . 95
5.2 Components for Context . . . 97
5.2.1 Context Manager . . . 97
5.2.2 Context Sensing . . . 98
5.2.3 Model Representation and Access . . . 102
5.2.4 Ontology Exploration . . . 103
5.2.5 Adaptation to Context . . . 105
5.3 Components for Application Generation . . . 107
5.3.1 Navigation Generation . . . 107
5.3.2 Content Generation . . . 111
5.3.3 Service Incorporation . . . 112
5.3.4 View Filtering . . . 113
5.3.5 Presentation Information Generation . . . 113
5.3.6 Summary . . . 113
5.4 Context Simulation . . . 115
5.5 Case Study: Example Web Application . . . 115
5.5.1 Application Generation . . . 115
5.5.2 Adaptation Effects . . . 116
5.5.3 Context Simulation . . . 121
5.6 Discussion . . . 122

6 Summary and Future Directions **125**
6.1 Summary 125
6.2 Future Directions 127

A Standards and Standard Authorities **129**

B Model Elements Type Definition in OWL **131**

C Models for a Prototypical Web Application **137**

D CATWALK Framework UML Diagrams **145**

List of Figures **149**

List of Tables **151**

List of Listings **153**

Bibliography **155**

List of Abbreviations

API	Application Programming Interface
CASE	Computer-assisted System Engineering
CBSE	Component-based Software Engineering
CMS	Content Management System
CSS	Cascading Style Sheets
CORBA	Common Object Request Broker Architecture
DOM	Document Object Model
DTD	Document Type Definition
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSR	Java Specification Request
J2EE	Java 2 Platform, Enterprise Edition
OWL	Web Ontology Language
PDA	Personal Digital Assistant
RDF	Resource Definition Framework
RSS	RDF Site Summary
SoC	Separation of Concerns
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAP	Wireless Application Protocol
WML	Wireless Markup Language
WSDL	Web Services Description Language
WSRP	Web Services for Remote Portlets Specification
W3C	World Wide Web Consortium
XHTML	Extensible HyperText Markup Language
XML	eXtensible Markup Language
XPath	XML Path Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation
XSP	eXtended Server Page

Chapter 1

Introduction

1.1 Theme Explanation

In software, networking can be used to provide the user with up-to-date information and access to services offered by a remote provider. In novel applications, networking is increasingly handled via Web mechanisms (Rheingold, 2002); corresponding Web-based systems support in particular applications accessed in a Web browser on a personal computer, and software for mobile systems that operates over the Web.

Web-based systems are increasingly complex and used for a wide range of purposes, possibly in changing circumstances, for example in various locations and with different devices. This has raised new issues related to applying the technology, in particular regarding usability of such systems. The amount of information provided to users of Web-based systems is increasing rapidly; this "content explosion" (Monaco et al., 2001) can result in confusion as to which information is relevant. Within many current Web applications, too much cognitive effort is required by the user to "recognize important relationships and recall key information" (Griswold et al., 2003). Furthermore, though Web-based systems have traditionally focused on information delivery, they are now also used as a platform for providing additional types of services, which may require complex interactions with the user (that is, interactions beyond mere activation of a link in a Web page). This raises the question of which services to provide in which situation, and how to integrate as seamlessly as possible services within the offering of a Web-based system, to reduce the amount of effort a user must expend to find and use a service.

Web resources are increasingly plentiful, but a user's cognitive resources are not unlimited. This is likewise true of the physical resources a user has access to in a specific situation. No matter how much bandwidth and computing power increase, there will always be scenarios in which these resources are insufficient, in particular in mobile computing scenarios (Forman and Zahorjan, 1994). Web engineers must address these problems of cognitive overload and physical constraints, as they restrain usability of Web-based systems and may even cause users to entirely reject a system.

Adaptation has been proposed as an approach to address these problems. In such an approach, a system is conceived in a manner such that it is capable of adapting its operation according to the present situation of usage. Corresponding Web applications can assist the user in navigation, content and service selection according to known preferences and current circumstances: an application should present the user "the right thing at the right time in the right way" (Kappel et al., 2003). In addition, for wider acceptance in everyday usage, an application should "require human attention for only critical aspects of task execution that require their input" (Thayer and Steenkiste, 2003). Thus,

a requirement for software can be that software should adapt its offering according to the situation; meaning that what is being offered, and how, should depend on the user and the *context* in which the user is using the system (Morse et al., 2000).

1.1.1 Adaptivity and Context

Varying terminology is used in computer science to describe the goals of adapting software to user needs, and of providing the conditions necessary for this adaptation. One needs to distinguish, on the one hand, the activity where a system engineer, or an individual user, explicitly and manually adapts an application to needs and preferences, and, on the other hand, automatic adaptation of an application's behavior, performed by a system.

Software supporting user-initiated individualization can be called adaptable, whereas software supporting system-initiated individualization can be called adaptive (Oppermann, 1994). A distinction in adaptability can be made according to the person performing the individualization. When the end-user tailors an application according to preferences, this activity can be called personalization. When a manufacturer configures or modifies software according to customer requirements, this can be called customization. When a customization is aimed at a large number and variety of customers, the term mass customization (Gilmore and Pine, 2000) can be used.

In this dissertation, we call a system adaptive when it is capable of automatically adapting an application's behavior, during application usage, according to the context in which the application is being used, that is, the "context of the interaction" (Dix et al., 2003). Additional terminology is sometimes used in research. The field of adaptive hypermedia uses the term personalization to describe the activity of systems which analyze the user's past behavior in order to decide what information to provide next, for example in tutoring systems (Brusilovsky et al., 1998). In mobile computing, the term customization is used to describe the goal of software adaptation to a user's location of use and device constraints (Badrinath et al., 2000); ubiquitous computing likewise uses this term in this sense (Kappel et al., 2003). Using the term customization for automatic adaptation can be misleading, as customization can also be used to describe an engineering activity, as discussed above.

To possess the ability to fulfill an individual user's needs, software must take into account several factors, such as the user's profile, current task or goal, and possibly additional factors such as location, time, or device used. The combination of all relevant factors in software usage can be termed the context, and thus a Web software that takes them into account is a context-aware application. The term context-sensitive is sometimes used interchangeably, see, for example, (Clerckx et al., 2005). In turn, the term context-adaptation is sometimes used to refer to the activity of systems basing their adaptation on context (Mani et al., 2004); this term can however be misleading, as it might also refer to the activity where a system adapts its knowledge about context itself, as opposed to adapting the application according to this context. To use information about context, a system must determine this information during its operation: this activity is called context sensing, context reasoning, or context inference.

In summary, using the notion of context in software (that is, "context-aware computing") is a paradigm in which information about the situation of system usage is gathered and structured in such a way that this information can be used for (automatic) adaptation of software according to the situation. The motivation for context-awareness (and with it, adaptation) is, ultimately, to increase usability of systems. In an adaptive Web application, knowledge about context might be used at various levels, from the structuring and appearance of the user interface to the information and services provided.

1.1.2 Web Applications

A Web application is software which enables a person, via a user interface, to make use of information and service offerings provided on the World Wide Web (in short, Web). To describe such applications, a definition of the Web is then required; according to the Merriam-Webster Dictionary, the World Wide Web is:

> a part of the Internet designed to allow easier navigation of the network through the use of graphical user interfaces and hypertext links between different addresses – called also Web.

The purpose of a Web application is traditionally to provide users with an access point to an information system, respectively to several information systems integrated for external (Web) access at this point (Rheingold, 2002). Increasingly, these applications go beyond mere presentation of information, and provide many types of services, sometimes requiring complex interaction: in other words, these applications provide operational features in addition to information delivery.

This dissertation focuses on software designed for the Web in a professional environment. In this regard, a Web application belongs to a specific sort of network-centric software systems used between an enterprise and its customers, or within an enterprise. Using Web mechanisms for networking can be compared to other fundamental approaches as outlined in Table 1.1.

	Client type	*Service offering*
host	terminal	A user is directly connected to a host, via a terminal. The user chooses a specific transaction, and then interacts with this transaction.
client-server	custom	The user interacts with a client software that is located in the same network as a server. The client software has some knowledge of the application domain and relies on the server to provide functionality such as data persistence: for example, a client might be connected to a relational database management system (the server), and interact with several tables in the database.
Web	standard (Web browser, WML browser)	A Web server offers information and services at a specific location, identified via a URL, and Web clients communicate via the HTTP protocol with the Web server. The client has no knowledge of how the server implements its functionality.

Table 1.1: Network-centric applications in a professional environment

A significant particularity of Web mechanisms is that they are standardized: notably, the manner in which resources are addressed (URL[1]), and the way in which client and server communicate (HTTP[2]). Unlike a typical client-server application that requires a custom client and operates in a controlled environment, Web applications can be used by a general purpose Web client, such as a Web browser, by people not necessarily known to the service provider. The fashion in which a Web application is used is thus less predictable than in classic software scenarios (Dumke et al., 2003): though a Web application is often targeted at certain groups of users, the actual type of user and the usage scenarios can not be known precisely. This requirement, along with requirements following from

[1] Uniform Resource Locators, see Appendix A
[2] Hypertext Transfer Protocol, see Appendix A

the specification of the Web standards, results in specific constraints for Web applications, notably regarding user interaction with these systems.

The goal of a Web application as understood in our work is not to recreate or replace existing information systems. The information systems themselves (referred to as "backend" in this regard) need not be engineered using Web technology, but the user's access to these systems is increasingly occurring via Web technology. To illustrate the difference, consider the following scenario: the Web offering of a health insurance company provides Web access to medical knowledge that can be queried and answered by a medical expert system (the backend). This Web application integrates a possibility for users to obtain knowledge from this medical expert system, but does not replace the expert system.

1.1.3 Web Engineering

Web Engineering is an emerging discipline that advocates (Deshpande and Murugesan, 2001):

> (...) establishment and use of sound scientific, engineering and management principles and disciplined and systematic approaches to the successful development, deployment and maintenance of high quality Web-based systems and applications.

While related to Software Engineering, Web Engineering must take into account, in addition to the technical particularities of Web applications (see Section 1.1.2) specific characteristics of Web-based systems, notably:

- Web-based systems are primarily content-driven (although they may in addition provide operational features). Navigation through this content must be achievable in a user-friendly manner.
- "Multiplicity of user profiles" (Murugesan et al., 2001): Web-based systems are potentially used by people with differing skills, capabilities, and interests.
- Web-based systems can be used in varied operational environments. For example, a user might use the same system while in the office and while performing external work, potentially with different devices.
- As a Web application can be a complex, multi-layered system, people involved in development can be of varied backgrounds and areas of expertise (Ginige and Murugesan, 2001), for example, domain experts, graphical designers, marketing professionals, software developers, or database administrators.

To gain an understanding of the increasingly complex requirements within Web Engineering, it is helpful to consider the evolution of Web applications. This evolution is typically considered in terms of generations, of which one possible categorization, as given in (Dumke et al., 2003), is:

- First generation: information is published on the Internet, and can be viewed by any user with a Web browser.
- Second generation: the user can interact with server components that provide complex information services, for example by integration with a database. Such a database may have existed prior to Web application development, and its contents are then made accessible through the Web.
- Third generation: plug-ins for the display of special media types in the browser become available.

- Fourth generation: pages are dynamically generated by the Web server, and the information provided by the system is managed consistently on the server using tools such as Content Management Systems (CMS).
- Fifth generation: this generation holds the goal of a "Semantic Web" (Berners-Lee et al., 2001), where association of meaning to information on the Web allows for smarter machine processing of this information.

The definition of generations of Web applications is arbitrary and varies in literature; (van Ossenbruggen et al., 2001) for instance use the term third generation to refer to the Semantic Web. Likewise, the distinction between second generation and third generation Web software - see, for example, (Copeland and Hwang, 1997) - is not always clearcut. In addition, individual applications, for example an Internet Banking software, can not be clearly assigned to one of the above generations. It can further be argued that development of software components to be used over the Internet (such as in automated B2B-scenarios), even if not used in a Web application per se, should also be considered Web Engineering; however, such scenarios are not within the scope of this dissertation.

An additional complication in categorizing applications is that, increasingly, applications tend to blur the boundaries between host, client-server and Web mechanisms. For instance, a client-server application can communicate via the same mechanisms as a Web application, or a Web browser can contain extensions that act as a client in the sense of a classical client-server application.

Nonetheless, specific requirements for state-of-the-art Web applications emerge:

- the information provided can be dynamically created and should be managed consistently;
- it should be possible to provide operational features within the Web application; meaning, functionality in addition to the classic Website operations browsing and navigation (Baresi et al., 2000);
- adaptation to context can be required (see Section 1.1.1);
- in accordance with the vision of a Semantic Web, applications should be based on semantic definitions of the resources provided by the applications.

Models and Methodologies

An approach to Web Engineering, as any approach to engineering, requires a description of the object to be engineered (in this case, a software application), that is, requires a *model*. In general terms, a model, according to the American Heritage Dictionary, is

> a schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics.

In informatics, the activity of modeling serves, according to the Merriam-Webster Dictionary, "to produce a representation or simulation of using a computer to model a problem." Within Web Engineering, modeling aims at producing a *representation* of the Web application; this application fulfills specific needs (that is, solves specific problems). Different existing modeling approaches greatly vary regarding scope of the problem that they address (examples for modeling approaches are provided in Chapter 2). Some approaches aim at supporting the design phase of application development; that is, at providing a common representation of the application to be agreed upon by all parties (for example, customer, designers and developers). In such approaches, the intention is that application developers

use the models as basis for their software programming task, whereas model usage by a run-time system for application generation is not aimed for. Other approaches propose to use a software system to interpret the application model in order to generate the application to some degree. The model can be considered a basis for *simulation* when directly usable for application prototyping. In the vision of generative programming (Czarnecki and Eisenecker, 2000), an application can be entirely described by successive models, thus foregoing the need for any custom software programming. Intermediate approaches can aim to reduce the need for programming or to allow for rapid prototyping, whereas a commercial-grade application would still require some form of additional, specific programming.

Application models thus have various scopes and purposes. For an individual engineering activity, it must be decided which sorts of models are to be used, what each model should address, and how and in which order they should be established. These questions are addressed in a *methodology*. Generally speaking, the purpose of a methodology in Web Engineering, as in software engineering, is then to support the software development process. The process is divided into phases aiming to better manage complexity and to ensure a higher quality of software. A methodology defines models to assist the Web engineer in development; such models also aim to facilitate communication between the various people involved in a development process. The degree of formalization of the models and how exactly the models are then used in development varies significantly between methodologies.

A model for a Web application must address precisely that which is relevant for the Web application at hand. We view a Web application as a user's unifying access point to one or more information systems (see Section 1.1.2). The Web application potentially uses and integrates systems which in turn are responsible for specific knowledge processing (for example accounting systems or expert systems). Consequently, modeling addresses precisely that which needs to be considered at the level of the Web application, and not at the level of a complete information system. Regarding the health insurance provider example given in Section 1.1.2, the following might then be modeled regarding the Web application: the existence of a medical expert system as a type of service, and the way in which the service can be used. Medical knowledge itself (such as, which symptoms are indicative of which sicknesses), which is contained in the expert system, is not modeled within the Web application environment. It may however be useful and necessary to share some sort of medical taxonomy between the systems in order to cooperate more meaningfully.

Information Representation

We now discuss a general way of representing information and relationships between information, applicable in particular in Web Engineering. This means of representation provides a basis for Web Engineering models and methodologies addressing the requirements discussed above.

The current W3C[3] standard to serve as basis for describing information is XML[4]. XML provides a common syntax for various purposes and is extensible, meaning new vocabularies can be created for new purposes while still conforming to XML. For an introduction to XML use and related technology within Web Engineering, see (Doernhoefer, 2004).

When a new vocabulary is to be used, it needs to be precisely described if it is to be widely adopted and if information described with this vocabulary is to be consistent (van der Vlist, 2002). A schema language can describe vocabulary and structuring of documents which should conform to a schema. Such schemas can then be used for documentation, validation and automation purposes. A classic language from the SGML[5] domain is DTD (Document Type Definition); other languages, such

[3] World Wide Web Consortium, see Appendix A

[4] The eXtensible Markup Language, see Appendix A

[5] Standard Generalized Markup Language, see Appendix A

as RELAX NG (van der Vlist, 2003) or Schematron (Lee and Chu, 2000), have also been proposed for formally describing XML documents. A DTD is adequate for describing the structuring of a document. Data type definition capabilities of DTDs are, however, limited. When richer data types are required, for instance, to specify that a document element refers to a URI[6] , XML Schema can be used. XML Schema is the W3C recommended language to define schemas for XML vocabularies; see the above mentioned (van der Vlist, 2002) for an introductory text. By its rich data type definition capabilities, XML Schema allows more possibilities in automation, for example, by using information about the types of parameters for a service in order to provide a more appropriate user interface.

XML thus defines a syntax for a vocabulary, and a schema language is used to define the vocabulary and the types of information represented. To be compatible with Semantic Web requirements, Web applications must include descriptions of the resources they provide and the relation between these resources. In related research, efforts for systematization are typically based on describing a conceptual model in terms of an *ontology* (Gómez-Pérez et al., 2004). This ontology represents terms and relations for a domain of applications, and can be called a *domain ontology* (Wissen and Ziegler, 2003). Such an ontology should describe the domain in a reusable manner, unlike a data model, which typically "represents the structure and integrity of the data elements of [a] single, specific enterprise application" (Spyns et al., 2002).

For formalizing ontologies describing Web resources, OWL[7] is the W3C's current recommendation. OWL builds upon RDF, the resource description framework (McBride, 2004), that allows to describe the resources involved and relationships between them. OWL adds possibilities in expression for ontologies as explained in (Antoniou and van Harmelen, 2004), notably cardinality restrictions and the assigning of special characteristics to properties. OWL, as is RDF, is an XML vocabulary, defined by an XML Schema.

1.2 Problems and Challenges

The general challenge existing in Web Engineering - see the above-cited (Deshpande and Murugesan, 2001) - is supplemented by the goal of accounting for context of usage on the one hand and, on the other hand, defining which adaptation is to occur (and how) according to this context.

Supporting adaptation to context in a systematic approach is a worthwhile effort, as the goal is to positively affect quality of Web-based systems, specifically usability and efficiency aspects as outlined in ISO 9126 (see Appendix A). Regarding usability, the goal of adapting and optimizing the application to context is to make the application more operable for the user. This is of particular interest in scenarios where the user has less time for an interaction or less powerful devices, such as in mobile scenarios: the goal is that the application becomes easier to understand and to learn. Optimizing the application in this regard can also positively influence Web efficiency, a further quality aspect. The tailored offering, made possible by context-awareness, decreases Web traffic, as less irrelevant information is exchanged. The user achieves goals more quickly, so less interaction with the system is required.

Attempting to integrate context-awareness and adaptation capabilities in a comprehensive approach poses several challenges. To effectively use context in Web applications, a formal, comprehensive model is required, as is a corresponding representation language. Context has a dynamic dimension and is therefore difficult to model. It must then be determined which elements of context are to be considered in the Web Engineering process, and which dynamic factors are added by a con-

[6] Uniform Resource Identifiers, see Appendix A
[7] Web Ontology Language, see Appendix A

text system. A conceptual model integrating a multitude of factors is needed, thus raising the question of how to structure and systemize the analysis and design processes.

In turn, aiming at a systematic approach to support context-awareness and apply adaptation within Web applications at run-time raises additional questions and challenges. How should information about context be specified in such a way that a run-time system supporting Web applications can use this information for adaptation, and what is an appropriate architecture for such a system? Since engineering approaches should support a multitude of scenarios, how can the effort required to develop a specific context-aware application be reduced? Furthermore, how can operational features of a Web application be specified, and user interaction with such services be supported in a systematic way, while taking into account requirements for adaptation to context?

A multitude of research projects and prototypes explore the context paradigm (this is illustrated in Section 2.1.1). Such research projects help to explore the sorts of context and possible adaptation effects; however, existing projects often explore a specific scenario and implement adaptation in this scenario only (Kappel et al., 2003). In addition, a (usually implicit) assumption exists in this field of research, namely that using context improves a system in some manner. Qualitative questions however are difficult to address at this time, as they require to systematically compare an "adaptive" application with a "non-adaptive" counterpart with similar goals and functionality. Current research prototypes are isolated applications that are not used over extended periods of time. These applications provide a specific functionality whose usage can not easily be compared with classic applications. What would be required to be able to address qualitative issues is that Web engineers have a systematic means to specify context information in Web applications, and to be able to activate and deactivate usage of context as desired.

1.3 Goals of this Dissertation

The main goal of this dissertation is to show how information about context can be taken into account in a Web Engineering process, that is, in a systematic approach to development of Web applications, and to show how adaptation effects according to this context can be achieved.

To enable the usage of context in Web Engineering at large, a multitude of application scenarios and context inference techniques need to be supported. A conceptual framework for context, using ontological descriptions, allows a uniform view on factors of context and their influence on application elements. The resulting context model is a basis for specifying knowledge about the context of an application and a basis for context inference algorithms. The engineering process must allow, in addition to context modeling, the specification of adaptation effects at each level of a Web application: navigation, resource selection and presentation.

To support adaptivity, a system architecture for Web applications faces particular requirements: support for context sensing and inference, and generation of a Web application where each phase in generation can depend on context. In the method presented in this dissertation, application and context models are designed to be interpretable by a run-time system, which should then be capable of using the models to guide application behavior. In particular, such applications must support operational features, potentially selected and operated according to context.

A software framework can facilitate the consistent usage of application models and generation techniques corresponding to an engineering method for adaptive applications. As various context inference mechanisms and adaptation algorithms may exist, the framework must be customizable and extensible for application developers. Furthermore, to enable testers to verify the effects of context on an application, the run-time environment should support testing individual factors of context.

Scope

This work makes a contribution to the discipline of Web Engineering, specifically to the issue of *engineering and development methodology and techniques*, by showing how context can be accounted for in the engineering process and what techniques can be used to achieve a context-aware, adaptive Web application. Other issues in Web Engineering, such as measurement, evaluation techniques or scalability are also essential this discipline, but not subject of this work.

The work presented here is rooted within the WISE research project (WISE, n.d.). The goal of that project is to provide a methodology and supporting tools for systematic development of complex and dynamic Web-based applications. The scope of this dissertation is to augment the WISE methodology to account for context and support adaptivity. Other project elements, not addressed in this dissertation, include the definition of a general-purpose methodology (see Section 2.3 for an introduction) and the development of an integrated, graphical modeling environment, including mechanisms for distributed development and version handling.

Summary

The main focus of this work is to show where and how context can apply in Web Engineering. By definition, the discipline of Web Engineering requires a systematic approach to Web application development, and accordingly usage of context within this discipline must be of a methodical nature. However, defining context in a general-purpose, yet precise manner is challenging, as is enabling consistent usage of context in an engineering process and a corresponding run-time system.

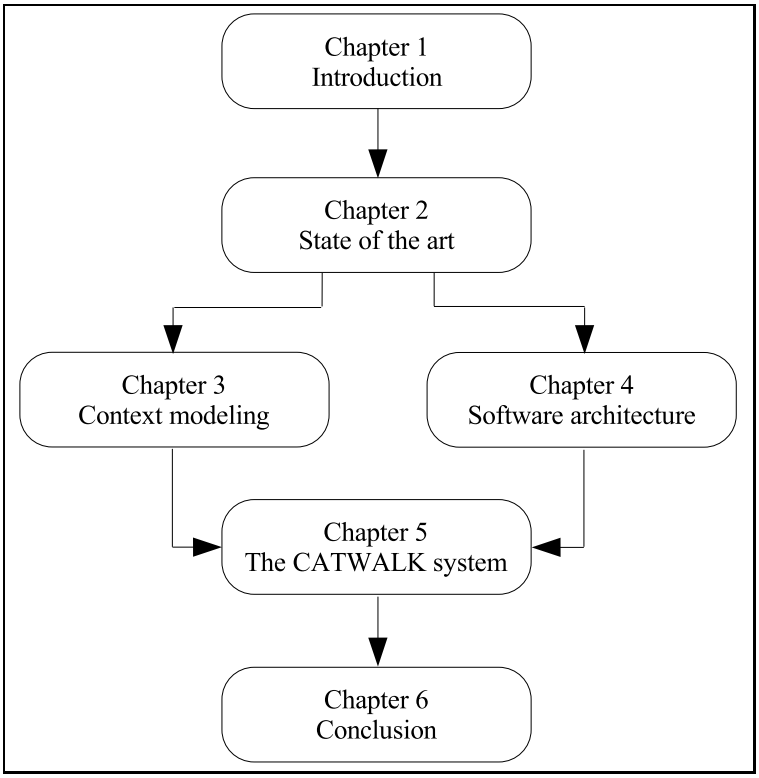

Figure 1.1: Chapter overview

This work makes a contribution to these goals by describing a conceptual framework for context

(Chapter 3). It is further discussed which adaptation effects can occur in an application and how these effects can be specified in relation to the definition of the core application. It is then shown how these specifications can be utilized by a content- and service-oriented system architecture to achieve adaptation to context in Web applications (Chapter 4). A software framework, facilitating the implementation of corresponding applications, is detailed in Chapter 5. The results and contributions of the dissertation are summarized in Chapter 6. Figure 1.1 recapitulates the chapters and their primary focus.

Chapter 2

State of the Art

In this chapter, we address the state of the art regarding the goal of engineering Web applications that can adapt to the context in which they are used. We begin by discussing context-aware computing in Section 2.1. General approaches to development of Web software are discussed in Section 2.2. In this area, a systematic engineering approach is often lacking (Barry and Lang, 2001), leading to problems such as difficult maintainability and limited re-use. A focus of research in the area of Web application development is thus to provide methodologies and tools for systematic engineering. We discuss general-purpose methodologies for modeling Web software (Section 2.3), and consider to what degree they support the goal of developing adaptive applications.

2.1 Context-awareness

Context-awareness, as understood in this work, aims to provide a system with knowledge of the context in which it is being used, such that system operation can adapt according to this context (see Section 1.1.1). In this section, we discuss current usage of context in software development and give an overview of applications in this area. Subsequently, we discuss aspects of particular interest to our work, namely architectural approaches to adaptivity, and the issue of what influences using context in a system can have on users.

Mobile and pervasive computing environments are examples of operational variations that Web Engineering needs to take into account. Because the constraints described in Section 1.1 are particularly visible in such environments, research in mobile computing introduced the notion of "context-aware computing" (Schilit and Theimer, 1994) to describe software capable of adapting to location of use.

A comprehensive context definition should be of a domain independent, non-excluding nature, and indeed several such definitions for what context means with regard to software have been proposed. (McCarthy, 1993) defines context as "a generalization of a collection of assumptions" and considers context as being of infinite dimension; see also (Brézillon, 2003). On an abstract level, context can be defined as "a focus of attention" (Brézillon, 1999). Even though a comprehensive abstract definition of context is necessary to pave a common base for all context-related research, the computer science field - as one of many - requires a more adjusted definition. An often-cited general definition is given by (Dey, 2001):

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

2.1.1 Accounting for Context

Varied research and application domains are using a notion of context to assist or guide the user; some domains employ the term context-awareness for this purpose, while others aim at achieving similar effects but use a differing terminology. A consensus on terminology does in fact not exist; see also Section 1.1.1.

Context-aware mobile computing research focuses on providing context-relevant information to mobile users, usually based on their location as context information. This notion has evolved to include other aspects, such as adapting the application to device characteristics (Schmidt, 2000). Nonetheless, a survey analyzing usage of context-awareness (Chen and Kotz, 2000) has shown that most research in this area has aimed in essence at sensing the location of a user in a mobile scenario, and using this information to adapt an application according to this location. A typical scenario is a touristic visit in which the user is guided by a mobile device such as a PDA; see, for example, (Cheverst et al., 2000). Research such as (Dix et al., 2000) does suggest that other types of context, such as a domain context, are also relevant to mobile applications, but does not explore this issue in detail. Generally speaking, this field of research emphasizes "context sensing". The remaining aspects are sometimes unclear: "the design criteria of some of the approaches as well as the detailed specifications are unfortunately not available to the public" (Strang and Linnhoff-Popien, 2004).

Adaptive hypermedia applications have typically focused on the areas of learning and information retrieval, with the goal of guiding user navigation and presenting relevant information. An overview of this field of research is given in (Brusilovsky et al., 1998). This field aims to adapt information offering according to a user model, where this model is generally constructed from user behavior. (Liu and Maes, 2004) for instance describes how an individual's attitudes can be computed and thus represented within a user model. The tracking of user behavior over a period of time can be used to automatically determine a user profile: in (Carreira et al., 2004), a set of user profiles is defined, and the system delivering the application monitors which Web pages are selected by the user and what content is in these pages. The user profile is determined based on the selected content, and this profile is then used to propose further pages of interest. Adaptive hypermedia usually takes into account user categories, sometimes also types of tasks (Koch, 2001; Staff, 1997). This field of research typically does not explore further types of context, nor further issues regarding interactivity (beyond link activation) and service incorporation into applications.

A further field of applications is Web-based e-commerce systems, specifically B2C (business-to-consumer) and intra-B (e-commerce offering within a company) systems. In such systems, a consistent use of the Web by users allows for tracking user actions, for example, recently visited items and past purchases. This tracking thus establishes a user profile that can be implicit or explicit in such a system. Commerce evolves from a mass-product and anonymous approach towards a personalized offering, commercials and customer contact. Such system behavior can also be termed adaptive.

Table 2.1 recapitulates the domains using context-awareness and provides additional examples.

Table 2.1: Applications using context-awareness

Application domain	*Examples*
electronic tourist guides	(Cheverst et al., 2000) describe tourist visits guided by a mobile device. The device uses location information to present the visitor with relevant information.

Table 2.1: (continued)

personal secretary application	A "memory aid" (Lamming and Flynn, 1994) can store information along with contextual knowledge, meaning the context in which an event occurred, such as location, or task being carried out at the time. The goal is to support information retrieval at a later time, thus explicitly addressing the problem of information overload.
meeting scenarios	(Dey et al., 2004) implement the aCAPpella system that uses various clues to (automatically) determine when a meeting is occurring; when this context is determined, the system automatically launches a notes-taking application and loads the most recently used presentation file.
groupware	Context, as support for a concept of "awareness", is considered a key element to effective groupware (Brézillon et al., 2004).
office scenario	The Smart Doorplate application gathers location and activity information (as context information) about co-workers, and makes predictions about what an individual is doing, for example "on the telephone" and "free in 5 minutes" (Trumler et al., 2003).
education	Adaptive hypermedia for tutoring consider user behavior in order to decide what to do next. Typically, a user model is constructed based on user interaction with the system (Boyle and Teh, 1993); depending on context (the current information being visited plus the user model), the next pieces of information are chosen. In Semaphore (Burrell et al., 2000), the Web links relevant to a lecture are associated to a location; mobile users, when using the Semaphore client at this location near lecture time, are presented with these links.
operational documentation	In (Ramu et al., 2004), context consists of conditions, goals and resources and is used to link documentation with operations in the aviation field.
computer-assisted design (CAD)	(Hernadvölgyi et al., 2004) show how a 3-D shape can have a semantic context in the CAD system. This information can then be used to assist the user in designing the end product, for example to assist in placement of objects.

Table 2.1: (continued)

performance optimization	Context information (such as user type or current task) can be used in order to predict what the user is likely to do next and prepare resources that will be needed: this can for example be used for smart caching in information retrieval, or hardware activation in anticipation for faster response time (Beigl et al., 2003). Network routing can be improved as described in (Gold and Mascolo, 2001).
Augmented-Reality	Augmented-Reality applications, notably task-oriented applications such as machine repair, use location and user profile context for repair support: repair instructions provided depend on sensing the operator's location, and can also be adapted to level of expertise (Feiner et al., 1993).
health-care systems	In (Jahnke et al., 2004), the system senses a patient's condition and monitors history of condition according to context such as patient activity, to enhance medical support.
retail outlets	The Metro Future Store in Rheinberg, Germany, can track product location and some measure of people's behavior, for example how long a potential customer assesses a product by holding it up. In accordance to this context, additional product information can be displayed, or, more generally speaking, advertisements in the store are adapted to context.
games	Game software or software-supported game devices such as Pokemon uses task-oriented context (Pagulayan et al., 2003).
community support	Applications such as a "friend finder" as described in (Schilit et al., 2003) take into account location and user type as context information.
desktop software	User interaction with desktop software can be influenced by context-awareness: (Louis and Shankar, 2004) describes how interaction with a calendar and appointment application is enhanced by including various context sources. The Aura project (Thayer and Steenkiste, 2003) supports document creation scenarios within an Office software suite, and considers context to be the set of documents needed as a group to meet a certain goal.

To use context for adaptation as illustrated above, it must be decided how context is represented, and what architecture to use for context processing and adaptivity. The following section (2.1.2) discusses approaches addressing the former; the latter is discussed in the subsequent section (2.1.3).

2.1.2 Context Representation

To use context, it must first be decided whether context is information that requires an explicit representation independent of programming code. In a methodology accounting for context (see Section 1.1.3), a means of representation for this context is required to provide parties involved in engineering with a common basis for description. Whether this specification translates to an explicit system representation at run-time, or is used by application developers for individual programming code decisions, is a separate issue.

Context may be represented with regard to objects of the physical world, or as an abstraction. Ubiquitous computing sometimes uses physical artifacts to define context, such as a smart phone or a smart table (Schmidt and Laerhoven, 2001). For instance, when sensing that "chair1" is occupied, "chair2" is occupied, the door in the room is closed, the system can infer that the room is in the context "meeting". In (Beigl et al., 2003), context is represented as the values of battery level, processor load, link quality and number of active devices, and this information is added to communication packets within a custom protocol for networking for ubiquitous computing. For (Burrell et al., 2000), context is the connection between Web links and a physical location, and the date and time range this connection is valid. In (Trumler et al., 2003), context is represented as the location of persons, stored in a distributed manner (in each "Smart Doorplate" device). (Louis and Shankar, 2004) represent context as system activity: "binary activity data from the keyboard, mouse, a motion detector, and a speech sensor". When the described calendar application generates a reminder, the user indicates whether the type of reminder (visual, auditive) was appropriate; the system stores whether the behavior was desired along with the context (the "activity data").

Alternatively to representing context as physical world attributes, abstractions may be used, or, as typically done in mobile computing, context can be a combination of abstract information (such as "user profile") and physical attributes. Abstract representations may take the form of project-specific, custom representations, or may use general-purpose representation mechanisms, such as ontologies.

Examples of custom representations include (Cannataro et al., 2002), where context is represented as a three-tuple called "adaptation space", the first element representing user behavior, the second environment, and the third technology. Adaptation is then specified as constraints applied to Web pages, where the constraints refer to the user's position in the adaptation space. (Cheverst et al., 2000) utilize "contextual information, such as location, display medium and user profile" and represent "location objects" in a hierarchical information model and associate a set of HTML pages to each location. In (Dey et al., 2004), the user demonstrates a certain desired adaptation behavior to the system, which at the same time "collects sensor data and detects events". The described system, aCAPpella, uses machine learning to learn what adaptation to perform in which situation (i.e., in which context), for this user. The example described is "when a meeting occurs, load the most recently used presentation file and launch a notes-taking application". Context here is then represented as (system-learned) patterns of behavior for some sensed situation. (Jahnke et al., 2004) uses GRAS (Kiesel et al., 1995), a graph-oriented system, as a platform for the context fact base: in this case, the graph itself is the representation of context, not merely a visualization. (Brézillon et al., 2004) discusses computer-supported cooperative work, and proposes to distinguish between an individual user's context and the group context. A user's context is represented as the virtual workspace of this user in the groupware, whereas the group context is represented as the virtual workspace for a project. Adaptive hypermedia projects represent context as a user model; for example, in (Boyle and Teh, 1993), the user model "represents the knowledge of a reader". This information is used to determine the level of detail to be presented to the user.

Context representation is increasingly handled via ontologies. In (Chen et al., 2004), context is

modeled as a "collection of ontologies for describing places, agents, events and their associated properties in an intelligent meeting room domain". (Ramu et al., 2004) represents context as a categorization of descriptors for document parts (pertaining to operational documentation); this categorization in turn is represented in an ontology, and the "documentary units [are] tagged with meta-data describing the context". (Hernadvölgyi et al., 2004) propose to represent "geometric context", to assist in CAD, as the definition of product parts and their spatial arrangements in an ontology: for example, in a car, geometric context of a fender is that it is "adjacent to hood". (Strang et al., 2003a) propose a "Context Ontology Language (...) to describe contextual facts and contextual interrelationships". These are described in ontologies, following a specific schema, and defined according to "common subconcepts of aspect, scale and context information". For example, "spatial distance" is an aspect, and "kilometer" or "meter" are scales. Context bindings link service parameters to aspects, the goal being that parameter values might be automatically determined from context.

An alternative to representing context specifically for service usage is to use built-in mechanisms of a specific service paradigm, such as in (Keidl and Kemper, 2004), who focus on W3C Web Services (Web Services are discussed in more detail in Section 2.2.3). Context is considered "information about clients and their environment that may be used by Web services to provide clients with a customized and personalized behavior" and is explicitly represented using the Web Service standards. Each type of context used is published in a UDDI repository, and may have an associated XML Schema. When a service is called, the SOAP header block is augmented with context information, allowing a framework (on the receiving side of the call) to extract this information and make it available to the service's environment.

Ontology representation is used, notably, to provide "explicit semantic representation" and facilitate knowledge sharing (Chen et al., 2004). However, this approach may also have drawbacks when compared to specific models, e.g., location models for mobile computing. (Becker and Nicklas, 2004) offer that spatial models for context representation are adequate when very fine granularity is desired, such as location management of mobile devices in cellular networks. The key requirement is that these models are scalable, and therefore their representation is "typically rather straight-forward". Again according to (Becker and Nicklas, 2004), contextual ontologies allow semantic description and reasoning, but tend to be used for smaller scales, in a specific environment. It is further proposed that a combined approach, with transitions between both types of context representations, is possible, and desirable. (Schilit et al., 2003) describe an additional problem when abstracting location information as made available by a positioning system in the form of user coordinates: locations can be organized in a hierarchy and thus a specific set of coordinates can be mapped to an element in such a hierarchy; however, a different hierarchy may be relevant depending on further factors regarding the user, such as the role of the user.

2.1.3 Architectures for Adaptivity

An architecture enabling adaptivity according to context must, on the one hand, define how knowledge about context is handled in the system, and, on the other hand, define how this knowledge is processed. For software systems at large, a possible distinction in architectural characteristics can be made between centralized and distributed handling, whereas some architectures are mixed. Table 2.2 provides examples of approaches used for context knowledge handling and usage of this context; these approaches are discussed in the following.

In a centralized approach to holding knowledge about context, context data is modeled in the application environment, and some of the decisions within the application are based on this information. (Chen et al., 2004) suggest an architecture with a central component called "context broker"

	centralized	distributed
Context knowledge	(Cannataro et al., 2002) (Chen et al., 2004) (Becker and Nicklas, 2004)	(Samulowitz et al., 2001) (Strang et al., 2003a) (Keidl and Kemper, 2004)
Context processing	(Cannataro et al., 2002) (Chen et al., 2004)	(Samulowitz et al., 2001) (Winograd, 2001) (Chen et al., 2004) (Strang et al., 2003a) (Keidl and Kemper, 2004)

Table 2.2: Architectural approaches in research regarding adaptivity

that "maintains a coherent model" of context. (Cannataro et al., 2002) present an integrated model for adaptive hypermedia, where adaptivity is specified in the application environment, integrating knowledge about user behavior and environment. (Becker and Nicklas, 2004) describe an architecture with a centralized context model, influenced by sensors of the physical world and by applications using this model. In this proposal, the same context model is usable by several applications running at the same time.

Distributing knowledge about context across a network means in essence to augment a network with intelligence: the network in itself holds knowledge about which information and which services are relevant in which context. This network approach is explored by (Samulowitz et al., 2001), who focus on providing context-relevant information and services through a smart network. The client, a mobile device, sends "context-aware-packets" to the network, to request a service conforming to some (pre-established) constraints. Intermediary network nodes route this data to services via "service access nodes" that are service proxies controlling the services available in the client's environment. Services register themselves with a service access node and state in what context they are appropriate. Within the network, nodes listen for such packets and respond to them, for example by presenting a relevant service to the client, which may then choose to use the service. The scenario presented is a mobile user attempting to discover a printer which may be used, and to which the user can have physical access from the present location. The area of application is then the field of pervasive services in environments with intermittent connectivity.

Having distributed context knowledge typically implies that processing of context is at least partially distributed as well. A distributed processing model is described as a collection of distributed software components and the means of orchestration of these components. Approaches to distributed processing include service-oriented architectures and agent-based systems. An example for a service-oriented architecture is the above mentioned (Samulowitz et al., 2001); further examples are (Strang et al., 2003a; Keidl and Kemper, 2004), who focus on W3C Web Services as service mechanism. An agent-based architecture is being followed, for example, in (Winograd, 2001). In this blackboard architecture, communications go through a centralized server that routes messages to those agents subscribed to this kind of message. Communication occurs using the "announce-listen style". For example, components that provide environment data, such as location sensing, periodically post events to the blackboard. Other components requiring this data can listen for these events.

Architectures based on centralized context knowledge such as the above-cited (Chen et al., 2004) require a central component for context processing. However, such an architecture may also include a form of distributed processing. (Chen et al., 2004) aim to support a pervasive computing environment with agents, as illustrated in the "intelligent meeting room system" scenario. The system senses a user

(physically) entering a room; the context broker updates the context model and informs subscribed agents, in this case the "projector agent". This agent then acquires appropriate slide show information, via the context broker, and sets up a presentation with this information.

Regarding architecture, "there is no panacea: each solution will be suited to some environments but not to others" (Winograd, 2001). This dissertation is rooted within Web Engineering, and thus considers applications operating in the Web's request-response mechanism as defined by the HTTP protocol. Communication between a user and the system is synchronous, therefore adaptivity is only required when a user request is made; a decentralized (distributed) architecture for the core system is thus not necessary. This however does not preclude the core architecture from delegating partial functionality to distributed components, for example for sensing the physical environment. Information from such a component can then be looked up pro-actively when a user request is being processed by the core architecture. This look-up can occur either by direct communication between the core and such a component or by indirect communication such as a blackboard. In both cases, the core architectural design is not affected.

2.2 Common Approaches to Web Application Development

A systematic approach to Web application design must be able to specify the parts that the application is to consist of, and the way in which the parts are to be assembled (Schneider and Han, 2004). However, the understanding of what a part is varies significantly between approaches. Moreover, some of these approaches focus on a specific system aspect, on a part of the development process, or on specific types of Web applications.

A general-purpose conceptual approach to software development is object-orientation, which aims to encapsulate software functionality into objects. Regarding Web software, object-orientation usually means that the functionality provided by a Web application is developed in an object-oriented programming language (for example, Java[1]), whereas content delivered to the user of an application is specified using some other means. Though Web system logic may rely on object-oriented languages and patterns, the use of such a general purpose language does not immediately offer help with regard to the Web Engineering process. Work such as (Neumann and Zdun, 2001) or (Zope Community, n.d.) follows the object-orientation idea to define a notion of "Web objects". These approaches correspond in essence to an object-oriented wrapping of HTTP's request-response mechanism, similarly, for example, to the Java Servlet API[2], while providing for richer typing of request and response objects.

CBSE, component-based software engineering - see, for example, (Heineman and Councill, 2001) - is an approach where encapsulation of objects is described in a coarser granularity than when individual objects are described. For instance, the goal of a CBSE process can be to provide "commercial-off-the-shelf" software components (Heineman and Ohlenbusch, 1999). Components in this sense might be incorporated in some manner within a Web application; however a component approach in the CBSE sense does not replace the need for designing the Web application per se.

In the following, we discuss common approaches used specifically to support systematic development of Web applications.

[1] Specified in JSR-901, see Appendix A

[2] Specified in JSR-53, see Appendix A

2.2.1 Portal Technology

In the portal approach, a Web application is composed by describing the user interface components that the portal consists of, using a window abstraction. A portal aims to integrate content from various, possibly remote, sources, into a single view consisting of individual windows. Selection of windows is handled by the end-user, who explicitly chooses to activate windows of interest (a default portal view is presented if the user has not made any choices). In other words, a portal is based on user interface components for a Web application.

JSR-168[3] is a standardization effort for portals implemented using Java technology. This specification provides a definition for components within portals; these components are called portlets. A portlet is designed to be a pluggable user interface component inside a Web application (Vieregger, 2003). The specification defines a way to aggregate user interfaces of several components (portlets) into a single (HTML) Web-page. WSRP[4] in turn defines a mechanism by which a distant portlet can be integrated into a portal, by calling the (remote) portlet as a Web Service; thus offering a standardization for integrating distributed systems into a common portal, provided that these systems define their user interface in terms of a portlet component. The specifications do not define a structuring of the content, nor how sharing, reuse, management and editing are to be handled. For illustration, JSR-168 defines an (optional) "edit" mode for a portlet, providing a view to editing the content shown in the portlet - this is however only a definition for the mechanism by which a user interface for editing would be triggered, and not a definition regarding how this editing could actually occur, nor of how the contents to be edited are structured. As a portlet is in essence a user interface component, there are no semantics associated with the different portlets. The goal of context-aware computing however, is to adapt behavior and appearance of a system component, in this case a portlet. The portlet specifications provide no means to influence the inner operation of a portlet, and thus provide no direct added-value to the goal of a Web Engineering approach in which context-awareness is taken into account. Nonetheless, portals can be used for integration with external, unrelated system parts. For example, an individual portlet can be the user interface to an external Web application or to a Content Management System (see below); this is an approach followed in commercial portal systems such as Oracle (Vandivier and Cox, 2001) or BEA portal server (Block et al., 2003).

In summary, portals define a user interface aggregation strategy in which personalization by the user is possible (and in fact intended), but where adaptivity (which does not require explicit configuration by the user) is not supported.

2.2.2 Web Content Management Systems

A Web Content Management System (CMS) is an application used for managing Websites and their content; the goal of such an application is to assist in systemizing the Web publishing process. A CMS is user- and content-focused, therefore terminology and composition strategies often refer to pieces of content as specified and aggregated by a CMS content author. Sometimes the term component is used to refer to a functional building block that can be integrated into the Website, for example, a component realizing electronic forum functionality. The term component may also refer to software building blocks of the CMS functionality in itself (for example, an component to support content editing).

In the view of a content author, the Web application is a collection of documents, each consisting of pieces of content or descriptions for content to be produced. Typical pieces of content are Web

[3] Portlet specification, see Appendix A

[4] Web Services for Remote Portlets Specification, see Appendix A

pages and file assets, such as office documents or images, that can then be referenced by the Web pages and used for display in different parts of the end-user view of the application (Robertson, 2004). An essential requirement of a CMS is that Web resources can be created, managed and edited in a user-friendly fashion. Depending on CMS functionality, the author view may also allow for enriching the Web application with components in the sense of software functionality, that is, to specify more complex functionality to be offered to Website visitors (for example, the above-mentioned forum component).

The content authoring activity in the CMS may itself be supported by a Web application, therefore the authoring application must be distinguished from the live application. The former is the mechanism with which users, in the role of editors, can manage content; the latter is the resulting Web application (or Website) that is provided to (potentially world-wide) users. The authoring application allows editors to specify which elements the resulting Website should contain, and how elements are organized within the site, for example, through a hierarchical menu. The content elements that are published then constitute the live Web application, or, more precisely, the content the Web application delivers to users.

The structuring of documents and document contents in the CMS are fixed. Although some CMS document types allow a dynamic content description such as RSS[5] for news feeds, application definitions such as where this content is to appear in the live application, or how retrieval of information is to occur, are fixed. Requirements of adaptation, however, mean that which piece of information is presented in which situation must be determined at run-time (according to context), and can not be fully known in advance. Existing CMS do not provide for such adaptation features, although partial adaptation to user characteristics may be available, for example automatic selection of the appropriate language version of a document in a multi-language site according to user device settings. Meta-data can be attached to resources, thus potentially allowing for adaptation features such as context-relevant selection according to semantic descriptions of the resource. However, how the information necessary for adaptation would be specified, and how context could be used throughout the Web application is not addressed by current CMS. In addition, a CMS would need to provide an editor-friendly means for describing context factors and desired adaptation effects; and furthermore would require an architecture for the live application using this knowledge for adaptation.

2.2.3 Web Services

A Web Service, as understood by the W3C (Booth et al., 2004), can realize arbitrary domain-specific functionality. A Web Service can be developed to implement new functionality or to provide access to an existing software, database management system, packaged component, or backend system. Web Services are the W3C candidate for realizing a service-oriented architecture (Erl, 2005), meaning an architecture defining a set of components that can be invoked (possibly from a remote location), and whose interface descriptions can be published. Web Services represent loosely coupled interactions and can thus integrate disparate software domains and bridge incompatible technology: in this sense, Web Services can be considered “enterprise application integration adapters” (Newcomer, 2002). The way the Web Service is to be implemented is not specified, and can therefore be accomplished by various existing server-side technology, such as J2EE or .NET (Eberhart and Fischer, 2003).

[5] RDF Site Summary, see Appendix A

Interacting with Web Services

Interaction with a Web Service occurs over a network interface (Bussler et al., 2002), whereas the software components realizing this interaction may be on separate machines or not. Interaction in this sense may or may not be directly related to an interaction with a (human) user. The requirements of Web Services have lead to the specification of several XML-based standards, notably SOAP[6] and WSDL[7]. These specifications aim at defining how interaction with a Web Service is to occur, with regard to structuring information and on specifying how the service can be used.

Research concerning interaction with Web Services mostly focuses on involving Web Services in a choreography to support business processes, or on supporting automated sequences, where several Web Services interact in a sequence in order to provide a new service. (Zimmermann et al., 2005) is an example of the former; the latter is used for example to chain e-commerce transactions in B2B[8] scenarios (Terai et al., 2003; Maamar et al., 2004). While SOAP and WSDL define base mechanisms for interactions, the specifications lack elements necessary for defining interactions in a sequence. Additional mechanisms are therefore required; (Frolund and Govindarajan, 2003) for example proposes a ”conversation definition language” where a chain of Web Service interactions can be specified.

While Web Services indeed need to support automated interaction when used in a chain, the issue of supporting user interaction with, and possibly automating user consumption of Web Services is as yet rarely addressed in research. Web Services, as a component model capable of representing any sort of dynamic functionality, can be used to define operational features of a Web application. The question is then how services in this sense can be incorporated within a Web application, and what effect context-awareness may have on this integration. A Web Service as a component is relevant for the Web Engineering process as described in this dissertation if usable by a Web application in order to provide a direct use for the end user; in other words, if it is useful for the Web application to act as consumer of this service. This usage may involve (explicit) user interaction, while the Web Service may or may not be explicitly presentation-oriented in its definition. To support this interaction, a user interface must be provided: for example, an XHTML form in a Web browser which allows the user to enter necessary parameters for the Web service and to call it. In (Pierce et al., 2002), XML Schema types used in the WSDL definition of a Web Service are mapped into “visual widgets” (such as XHTML forms), which are implemented as Java Server Page templates, thus allowing for automatic user interface generation.

Another perspective on integrating existing dynamic functionality into a common Web application is to define mechanisms for aggregating existing Web applications (notably, applications offering no means to be used without user interaction in a Web browser) and supporting their automated usage. (Chiu et al., 2003) propose a script language called WebXcript to automate filling and submission of HTML forms to known Web sites. Here, a “web service” is an existing Web application available as a HTML interface, and not a Web Service in the W3C sense. The goal is to support dialog automation by augmenting HTML form technology with knowledge of the service to be interacted with. This approach has the advantage of allowing integration of offerings currently existing on the Web, however results in a tight binding between HTML and the services, and thus carries the problem of mixing of content and style: in other words, this violates the “separation of concerns” design principle (Fielding and Taylor, 2002). A service in this sense is not reusable for other GUIs (for example, WML[9] user interfaces). An alternative would be to develop a Web Service as a proxy to such an

[6] Simple Object Access Protocol, see Appendix A

[7] Web Services Description Language, see Appendix A

[8] Business-to-Business

[9] Wireless Markup Language, a document format for mobile phones

offering, to be removed later if the offering becomes available directly as a Web Service.

Research projects aiming at integration of context-awareness with Web Services include CoOL (Strang et al., 2003a) and (Keidl and Kemper, 2004). CoOL proposes a model describing "context bindings" and "context obligations" for services. In a WSDL description, these descriptions translate into context attributes belonging to a certain "aspect", which are elements of a catalog proposed in this model: for example, "Geographic Place Aspect" or "Weather Aspect".

A context processing architecture explicitly focusing on Web Services is introduced by (Keidl and Kemper, 2004). This architecture explores several possibilities for service integration: context pre- and post-processing around Web Services for automatic context processing, a notion of "context services" that implement a special WSDL interface, and a SOAP extension to specify context information within the SOAP header. The SOAP extension augments standard Web Service calls with context data and is thus usable by those Web Services that are context-aware while being transparent to Web Services that are not context-aware. That work provides insight for combing context and Web Services, however the question of integration within a general Web Engineering process is not addressed.

Further Service Approaches

The definition of a Web Service in the W3C sense being recent, one needs to be careful to avoid confusion with earlier or parallel work using this term in a different meaning, see, for example, (Brabrand et al., 1999). Other research projects refer to "Web services" to mean a service on the Web in their particular platform (Böttger et al., 2006), or as a general notion. For example, (Sun and Sauvola, 2003) describe an abstract vision of "context-aware adaptive services", however do not pursue a specific service mechanism. In this dissertation, the term "Web Service" refers to the W3C definition.

For providing services on the Internet, W3C's Web Services is the most recent attempt at standardization. Another standardization effort for providing services on a network is CORBA[10], defined by the Object Management Group (OMG). Web Services in fact share many common concepts with CORBA, such as an interface definition language, a service repository and dynamic service discovery. As stated by the OMG,

> Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.

Thus, CORBA proposes a protocol which can be used on various types of networks, including the Internet. Web Services instead focus specifically on the Internet and existing Internet protocols (such as HTTP), and use XML vocabularies for all definitions. For detailed comparisons between CORBA and Web Services, see (Gokhale et al., 2002) or (Baker, 2002).

ebXML[11] is a specification for business processes for Internet Commerce based on XML, defining a business process as a series of interactions among interested trading partners (Newcomer, 2002). It is intended for B2B (business-to-business) scenarios, as a replacement for classic EDI[12]-based applications. Some aspects of ebXML converge with Web Service specifications, notably regarding the transport layer, as the current specification of ebXML now uses Web Services' SOAP. As this

[10] Common Object Request Broker Architecture, see Appendix A. For an overview, see (Mowbray and Ruh, 1997)

[11] Electronic Business using eXtensible Markup Language

[12] Electronic Data Interchange

dissertation focuses on general-purpose *user* interactions with services, Web Services' WSDL and UDDI[13] standards have more immediate relevance.

Within mobile computing, concerns have been expressed regarding performance of existing service environments. The Vinci project (Agrawal et al., 2001), for example, aiming to support PDA scenarios, implements a custom protocol to achieve faster communication than SOAP-based messages.

Web Standards

As illustrated by the standardization efforts (such as XML and Web Services) discussed above, work in Web Engineering can involve the use of a multitude of standards. Existing standards are at varied maturity levels, and are maintained by various standard bodies. Appendix A provides an overview of standards relevant to this work, as well as their defining organizations.

An advantage of relying on standards is that, in many cases, software components supporting these standards are already available and can be used as building blocks. An additional aim of basing work on existing or emerging standards is to increase continuity of work despite the rapidly changing nature of Web Engineering.

2.2.4 Application Frameworks

The purpose of an application framework is to support application development by providing commonly needed features, and thus reducing the need for programming. A Web application framework, in particular, possesses built-in features to support requirements frequently encountered for Web applications, for example multilingual capabilities or integration of databases (Moczar and Aston, 2002).

A multitude of frameworks for Web applications exists, for example Jakarta Struts (The Apache Software Foundatio n.d.c) or Apache Cocoon (The Apache Software Foundation, n.d.a). In this section, we focus on Cocoon, as a framework explicitly supporting Web applications based on XML representations. It is shown how Cocoon is used as a framework for existing research in context-aware Web applications. Chapter 4 explains in addition how Cocoon can be extended in our approach to realize, in effect, a framework for adaptive Web applications.

Cocoon Principles

Apache Cocoon is an XML-based Web-publishing system. By conception, Cocoon also provides an XML framework for other systems; as stated in (Mazzocchi, 2002), the system was conceived to be like a cocoon that allows new Web technology (such as XML and Java Servlets) to "incubate and grow stronger". Figure 2.1 shows how Cocoon functions as central element of an architecture that provides for multiple channels, meaning a variety of user devices or usage scenarios, while incorporating data from a variety of systems (Ziegeler and Langham, 2002).

Cocoon's architecture is built upon mechanisms for distinguishing between content, style and logic, and thus explicitly addresses the fact that, in Web Engineering, people with varied areas of expertise can be involved in the development process (see Section 1.1.3). Content is specified in XML, where this XML can consist of an arbitrary, custom structuring, or conform to a standardized XML dialect. Presentation of content is dynamic and may serve various kinds of clients (such as Web

[13] Universal Description Discovery and Integration, see Appendix A

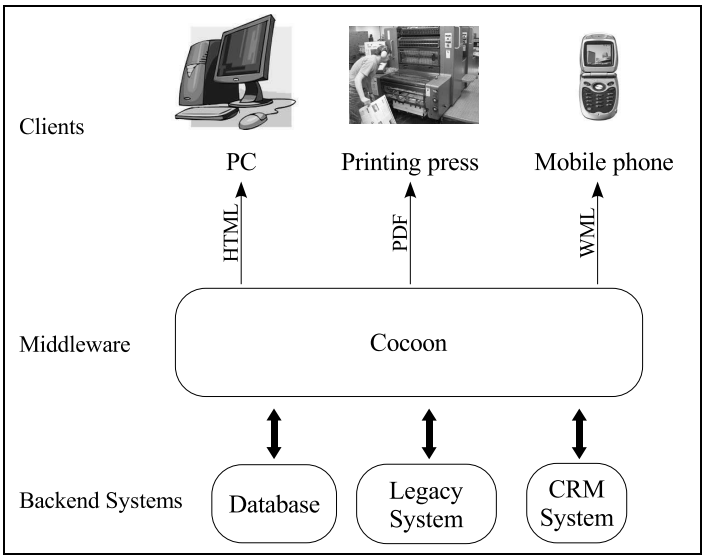

Figure 2.1: A three-tier architecture based on Cocoon

browsers or WAP phones) while being based on the same XML data: this can be implemented, for example, with XSLT[14] stylesheets.

XSLT is a subset of XSL[15], a language designed to assist in processing XML documents. For instance, XSLT can be used to generate a user-readable layout for an XML document, by transforming content into a presentable format, such as an XML dialect designed for display within a user client. XSL, specifically the subparts XSLT and XPath[16] are applicable not just at the immediate user interface level, but can be used for any XML processing step in an application: an input XML representation is transformed into another (XML or non-XML) representation, by rearranging input XML according to stylesheet transformation instructions that refer to incoming XML content.

The Cocoon sitemap allows to configure the sequencing of processing steps and the handling of state changes in the application (as resulting, for example, from user actions). Additional logic can be implemented as XSP (eXtended Server Pages), which are XML documents embedding logic-elements (fragments of Java code), or as custom components extending Cocoon. Such a component can be realized as one or more Java classes implementing certain Cocoon interfaces, and registered within the framework.

Research Projects Using Cocoon

Cocoon has been used in various research projects. Typical usages include the development of document management programs (Niinimaki et al., 2003) or multimedia applications (Bocconi and Nack, 2004; King et al., 2004). In a general software engineering perspective, (Bravenboer and Visser, 2004) cite Cocoon as an API[17] to generate XML that is guaranteed to be valid, within the context of syntactic descriptions for domain abstractions. In the component-based approach to software development, (Bandini et al., 2002) cite Cocoon as a "middleware service component" to be used, depending on identified "business components".

[14] XSL Transformations, see Appendix A
[15] Extensible Stylesheet Language, see Appendix A
[16] XML Path Language, see Appendix A
[17] Application Programming Interface

Cocoon is furthermore used to implement adaptive systems, notably for technology-based learning and mobile scenarios. Within the field of technology-based learning, the WHURLE prototype (Moore et al., 2001; Zakaria et al., 2003) models a learning process as a dialogue between student and system, in which (XML-encoded) information chunks are presented adaptively to student needs. The adaptation is implemented with the help of Cocoon's XML pipelines: such a pipeline is considered "the ideal place to implement a pluggable user model". Cocoon is considered an appropriate approach and technology to "leverage the techniques and technologies of adaptive hypermedia to deliver educational content that is appropriate to the needs of individual students". Quiver (Ellsworth et al., 2004) is another teaching aid system implemented with Cocoon, as is SSM (Darbhamulla and Lawhead, 2004), which aims at an "adaptive student interface".

In mobile computing, (Wegscheider et al., 2004) hold the premise that a frequent change of context means that a multi-modal user interface would be beneficial, and thus implement an interaction manager that relieves the application from having to adapt the user interface to specific devices and modalities. This manager builds upon the Cocoon framework. (Semrau and Kraiss, 2001) discuss mobile commerce scenarios based on the Wireless Application Protocol (WAP); these scenarios require to personalize the information with respect to user device limitations, notably display size and bandwidth limitations. To address these limitations, the system must decide "which information which user in which step of the process needs". Adaptation to various WAP devices is handled by an application-independent framework that uses Cocoon. Adaptation regarding user interface affects features such as image size, image inversion, alternative implementations of non-implemented commands, realization of homogeneous layout, and compensation of bugs. Cocoon's major contribution in this adaptation is the selection of the appropriate stylesheet, based on user agent properties. (Wilcock, 2003) integrates Natural Language Generation (NLG) with XML technology, using Cocoon: the system answers in natural language to enquiries such as "when does the next bus leave from stop X". Responses depend on "dialogue context", which influences, for example, the length of the response. Cocoon's support for reconfigurable pipelines of XSLT transformations is used to implement different pipelines supporting different NLG requirements.

Research projects furthermore recognize Cocoon explicitly as a valuable basis for constructing adaptive Web applications or for building frameworks to support adaptive applications. XVM (Li et al., 2004) is an application framework built upon Cocoon with the goal that XVM-based applications can *adapt* to the computing environment, for example, for handheld devices. This work also suggests that Web Services can be used to generate parts of (dynamic) XML documents. (de Andrade et al., 2004) describe an approach where XSLT and Cocoon are used to generate Web documents, focusing on the application as a collection of document parts. XAHM (Cannataro and Pugliese, 2001) aims to implement adaptive hypermedia based essentially on extrapolating a profile from user behavior (for example, links followed), and generates Cocoon XSP to implement the application server; the added value of this usage of Cocoon is however not elaborated upon. The AMACONT project (Fiala et al., 2004) is built upon the premise that for quick generation and delivery of information automatically adapted to user device and preferences, conventional document formats are not suitable as they lack support for the separation of content, structure and layout: this makes Cocoon an appropriate choice for a system that "enables to compose adaptive Web applications by the aggregation and linkage of reusable document components". AMACONT defines a custom XML document format for this purpose.

Work such as (de Andrade et al., 2004; Fiala et al., 2004) typically focuses on Web applications as document-delivery systems, and thus adaptation concerns document and document part selection and does not address operational features. The incorporation of operational features raises questions that are in some aspects similar to questions raised by the adaptive integration of document parts (such as

which part to present in which context), but are in other aspects different, consequently to the dynamic nature of operational features. Such features may require one, or several, steps of interaction with the user, for example, to supply missing information for the usage, and then, depending on the result, choosing the next step. Furthermore, the actual service realizing the feature might be dynamically discovered, thus requiring a more advanced integration mechanism.

Integrated Notions of Context in Cocoon

In Cocoon, and in the underlying software component framework with which Cocoon is developed, Apache Excalibur (The Apache Software Foundation, n.d.b), several notions of context already exist, that we now discuss to see if and how they are related to the understanding of context followed in this dissertation.

First, Cocoon, being a servlet-based application, runs within a servlet engine. This engine can run several different servlets at the same time, so the engine must differentiate between them: this is called *servlet context*. In essence, each servlet context represents a separate Web application.

The initial *Cocoon context* is the physical starting point, in the application server's application hierarchy, of the individual Cocoon-based Web application, represented by the main sitemap. As sub-sitemaps are mounted, Cocoon context shifts to these sitemaps. In this sense, the Cocoon context is merely a directory path. However, more interestingly, Cocoon provides the pseudo-protocols `cocoon:/` and `cocoon://`; these protocols can be used within a Cocoon-based application to access any resource belonging to the application. The former looks for resources in the current Cocoon context only, whereas the latter looks for resources beginning at the Cocoon root context. Resources are read according to the sitemap matching rules, meaning resources addressed in this way are served by the usual pipeline mechanism: as such, resources can either be files within the Web application archive (as specified in JSR-53), or can be dynamically generated.

Servlet and Cocoon contexts refer to mechanisms for handling the Web application within a server environment on the one hand, and on the other hand for providing the Web application with a way to access resources (whereas this resource can be dynamically generated). In this sense, these notions of context are more of a technical nature and do not directly relate to a general understanding of context in a Web application. The component framework (Excalibur) does, however, have an application-focused notion of context. A component may declare that it implements the `Contextualizable` interface; if it does, the component manager calls this component's `contextualizable()` method before the component is used. This provides a `Context` object to the component, that is a data store that can hold any information that the application wants to provide the component, using key-value pairs. While this provides a generic mechanism to pass context information to a component prior to usage, this does not define the organization nor meaning of this context information. The Excalibur project does state to be currently discussing specification of meta-information related to context and context entries.

In summary, the Excalibur and Cocoon application environment has several built-in notions of context that are of a rather technical nature and not to be confused with the user-centered notion of context our work focuses on; the environment furthermore provides a generic mechanism for passing context information to components. How such context information is structured and interpreted, and what is done with it, is not defined by this general purpose component framework. It should be noted in addition that Cocoon provides access to information that is considered context in this work, though Cocoon does not use this term for this information: notably user device parameters, user language settings, and session information.

2.3 Methodologies for Web Engineering

Succeeding generations of Web applications, with their increasing requirements regarding functionality (see Section 1.1.3), have lead to a multitude of proposals regarding systematic, structured development. In the following, we discuss selected models and methodologies that are representative for the goals outlined in our work; (Costagliola et al., 2002) provides a comprehensive survey.

Initial approaches, that can also be partly applicable to Web application development, originate in the field of hypertext application design. Typically, it is proposed to "specify the hypertext structure and behavior in terms of a well-defined conceptual schema" (Carr et al., 2001). In HDM, the Hypertext Design Model (Garzotto et al., 1993), a schema is used to represent the items and relations underlying the hypertext application, and a "set of access structures" on this schema define an application. The conceptual schema represents hierarchies of concepts and pieces of text, and relations between these hierarchies. Access structures can be defined to exploit the organization of the conceptual schema to specify links. Individual links in the hypertext can therefore be derived from the access structures, thus reducing the number of links that must be explicitly specified. Examples of access structures are "structural links" that can be used to define a navigation in a hierarchy, and "application links" which define navigation between hierarchies, based upon modeled relations between the hierarchies.

The Relationship Management Methodology (Isakowitz et al., 1995), RMM, proposes several access formalisms to define the navigation model of a Web site. RMM addresses "highly structured" sites, such as the Web site of a university department which lists faculty, courses and publications. To represent the information, an E-R[18] model is used, and entities can furthermore be subdivided in "slices", which are equivalent to the amount of information about the entity which is representable in a Web page. Navigation is then modeled based on this extended E-R model, and may contain links between slices, grouping constructs, and constructs representing an access on the information model: notably, an "index" representing a list of entity instances, and a "guided tour" representing a linear path through a collection of items.

WSDM, the Web Site Design Method (De Troyer and Leune, 1998), is specifically of interest for work considering context-awareness. In WSDM, user types and characteristics are defined, and a perspective on the site is defined for each user type. Such a perspective consists of a selection of elements from the navigation layer considered appropriate for this type of user, and is modeled in the "context layer". The issues of user type recognition and site generation are not addressed.

The models and methodologies described above specify composition of document resources, and are therefore appropriate for building systems which are "document-based" (Dumke et al., 2003). For systems requiring operational features, additional constructs and tools are necessary. Likewise, a system which initially does not require operational features may grow beyond initial intentions, so that the system needs to fulfill such additional requirements. The question is then to what degree a methodology supports this evolution, or if a patchwork approach, disconnected from the initial artifacts of the methodology, becomes necessary. Initial CASE[19] tools, as do the methodologies, focus on document-oriented Web systems. A commercial tool such as Dreamweaver (Watrall, 2002), offering partial CASE functionality specifically for Web sites, does allow to integrate dynamic components such as Java Applets (Schildt, 2004) or Active-X connections (Mueller, 1997). However, such items are technology-specific additions to the Web client; the tool has no knowledge of the actual purpose of this component within the Web system, nor about the component's relationship with the application

[18] Entity-Relationship, see (Chen, 1976)

[19] CASE: Computer-Aided System Engineering

domain or with other components. To support operational features, a methodology must address the specification of dynamic components of an application, their relation to other application parts, and the means and consequences of user interaction with these components at run-time. In eW3DT, the extended World Wide Web Design Technique (Scharl, 2000), components such as database connections can be represented in the design stage. Such a representation can then be a basis to implement an operational feature, for example the storing of information provided by a user via an HTML form into a database table. eW3DT addresses the design stage of a development activity: the models are not used by the run-time system, instead, it is intended that the system be implemented by programmers according to this design. This is also the case for UML-based methodologies such as WAE (Conallen, 2002) and UWE (Koch, 2001).

Model-driven Development

The goal of guiding an application's behavior directly by models is not accounted for by the above-mentioned methodologies, which address the requirements analysis and design phases of application development. Two general directions exist for using models to drive an application: code generation and model interpretation by a run-time system. In a code-generation approach, models are read by a code generator and transformed into a target language such as Java; this code can then be extended by programmers and compiled into the actual application (Rutherford and Wolf, 2003). Similarly, (Kraus and Koch, 2002) builds upon UWE to provide a "semi-automatic transition from [UWE] design models of a Web application to a running implementation". Apache Cocoon is extended and used to read the models resulting from an UWE engineering process; these models are provided in XMI, the standardized XML format to interchange UML models. Cocoon is then used to generate templates for logic (as Java class skeletons) and for presentation (as XSLT stylesheets). Subsequently, these templates can be used by programmers and Web designers to complete the application. In this process, "the use of UWE [is limited] to non-adaptive applications".

In approaches to model interpretation, a system must be developed that is capable of accepting user requests and generating responses (for example, in form of HTML pages) based upon knowledge described in the models. The system itself is not generated, but can instead interpret models for various applications; in other words, applications are generated at run-time by a supporting system.

The support for model interpretation at run-time is further complicated by requirements of context-awareness and adaptation within Web applications (see Section 2.1). In WebML, the Web Modeling Language (Ceri et al., 2000), an application's pages and navigation are described in XML. A corresponding system is described, that uses this XML to generate application code as Active Server Page (Mitchell, 2000) templates, which in turn produce HTML in the run-time application. The use of XML as a basis does however not yet answer the question of how context knowledge would be represented and how it influences an application, nor how operational features should be specified and integrated. (Manolescu et al., 2005) proposes to extend WebML to allow for Web Service specification, thus allowing to represent operational features via these Web Services. Further, WebML has some support for accounting for context as described in (Ceri et al., 2003), where individual pages can be marked as being relevant for a certain location (context); the example discussed is a museum application to assist guided tours with information delivered to mobile devices, according to the physical location of the user. Further aspects of context and context-aware integration of operational features within the Web application are not addressed. WebML uses an E-R model for conceptual modeling, and specializes on the use of relational databases.

Ontology-based Modeling

An alternative approach at modeling in a manner such that the model can support run-time applications is to base application modeling on an ontology. A domain ontology represents the terms and relations of the domain relevant to an application (see Section 1.1.3). The goal of ontology-based application modeling is to use knowledge thus represented to specify and/or generate elements of the Web application, notably the navigational structures; that is, the application is built upon ontology-based navigation (Crampes and Ranwez, 2000).

In the OntoWebber proposal (Jin et al., 2001), a navigation model describes how pages and "cards" (containers which can be aggregated in a page) are connected through links. Notably, a "list card" can represent a list of instances of a class in the ontology, and a "query card" represents a search for ontology entries satisfying a criteria. The focus of OntoWebber is on the integration of data from various ontologies, by proposing to define a reference ontology (which is then the domain ontology of the Web site) and to define mappings between the external ontologies and the reference ontology. Direct usage of the models for Web site generation is intended, but not provided.

The SHDM method (Lima and Schwabe, 2003) transposes an existing object-oriented approach, OOHDM (Schwabe and Rossi, 1998), to ontology concepts. The navigation model is created by specifying RQL queries (Karvounarakis et al., 2002), determining which elements of the conceptual model are retrieved, with which conditions. The conceptual model is an ontology, which may contain hierarchies of classes and relations between classes; that is, the level of expressiveness is the same as for the object-oriented model. Adaptivity is not addressed in SHDM.

Hera (Vdovjak et al., 2003) is a methodology that supports the engineering of "Web Information Systems" based on ontological descriptions of data in RDF. A Web Information System is distinguished from a Web application, and defined as a system which "generates a hypermedia presentation for data that in response to a user query is retrieved from the data storage": for example, a query regarding a painting results in an image and description of this painting to be retrieved. A particular focus of Hera is on data retrieval and integration from multiple sources, as in OntoWebber. The approach in Hera is to require the external sources to be accessible via RQL queries; the modeler links the external ontology concepts in a "meta-ontology". In (Frasincar et al., 2005), initial work for a presentation generator for Hera is described. The generator "supports the automatic building of (static) Web Information Systems from a given set of specifications (models)", and is triggered by the system designer. The generation process uses a pipeline of XSLT stylesheets to transform the RDF data into a presentation in HTML, WML, or SMIL, depending on the browser type. An adaptation model defines appearance conditions for pieces of information, based on preferences defined in a user profile. The user profile must be known prior to generation of the site, to steer the generation process.

OntoWeaver (Lei et al., 2003) is an example for a methodology proposing to represent all application models as ontologies, such that adaptation (called customization in that work) can be specified by rules based on entries of these ontologies. The models are intended to be automatically interpretable by a run-time system, however a corresponding system is not provided.

WISE, discussed in the following, is a project for a new methodology founded on ontology-based modeling.

WISE

The WISE project, within which this dissertation is rooted (see Section 1.3), aims to establish and support a general-purpose methodology for Web applications, in which a prototype application can be generated from models. The methodology recommends the following modeling phases (Wissen,

2004a):

- The domain ontology is modeled, resulting in a *conceptual model*.
- Navigational structures and definitions for composition of individual Website elements are established, resulting in a *navigation model*. Entries in this model are based on the previously established conceptual model, and can be references to individual ontology entries or represent dynamic accesses, for example, the list of instances of a class in the ontology.
- The *view model* is defined as an extension of the navigational model, to determine which Website elements are to be visible depending on the type of user. This represents one aspect of context modeling as understood in this dissertation.
- The last modeling phase details the graphical presentation of the Website. This *presentation model* provides style definitions for structures and elements established in the other models.

In WISE, the information base is thus described in an ontology. Navigational classes enable "classification of information objects in individual subjects of the navigation structure" (Wissen and Ziegler, 2003). Ontology exploration can be used to generate navigation possibilities, for example by specifying that the target of a link is a container for instances of a class defined in the ontology. Upon link activation, all currently existing instances are then displayed in an overview. The possibility to use any ontological knowledge to define a navigation extends classical hypermedia approaches such as CREF as described in (Conklin, 1987), where pages can be dynamically generated via the definition of a predicate over keywords associated with chunks of text. In addition to the resulting power of expression, the usage of ontologies can be motivated by the goal of interoperability between information systems providing an ontological description of the contained information, as recommended by the W3C in its Semantic Web efforts (Meersman, 2002).

2.4 Engineering for Context-awareness and Usability

Context-awareness, initially used to support scenarios for mobile software adapting to location of usage, is now used in a variety of scenarios (Section 2.1). To support adaptivity in applications, various architectures for context exist, depending on the exact system environment or purpose. Current approaches to development of Web applications can facilitate the development process, however in these approaches there is little to no built-in support for adaptation to context (Section 2.2). Methodologies for Web Engineering aim to systemize the development process, however existing methodologies (Section 2.3) insufficiently address all potential requirements of a Web application: notably, the need for adaptation to context and for operational features. The limited support for the requirement of adaptation to context raises two questions: how can a methodology be developed (or extended) to account for context and to support adaptation in Web applications, and in which situations is adaptation to context in fact a sensible requirement? The first question is addressed in the remainder of this dissertation.

Regarding the second question, it should first be noted that the goal of introducing context-awareness within software systems must be to improve this system in some manner, although in what respect the system is expected to be improved by using context-awareness is usually not explicitly addressed in current research. It can be surmised that most work regarding context-awareness aims either at influencing user choices regarding how the software is used and what actions the user takes as a consequence, or at improving system quality. A marketing-steered recommendation system is an

example of a system aiming at user influencing (Riedl and Konstan, 2002). Improving system quality, in turn, can mean to improve either system *usability*, which measures the quality of the interaction between a user and the system (Seffah and Metzker, 2004); or system *performance*, such as reducing bandwidth needs. These goals can complement one another; if usability is improved, the user can be more efficient and thus consume less system resources; reciprocally if the system is more efficient, it is more responsive and thus has a positive influence on usability. This dissertation focuses on the effect context-awareness can have on user interaction at the immediate level.

Work aimed at improving usability should ultimately also aim at providing means of measuring whether context-awareness has actually improved the usability of a system, and to what degree. Research in context-awareness, being at an early stage of development, often does not consider this question at all. Individual projects such as Sycophant (Louis and Shankar, 2004) are beginning to address the issue of measuring usability. Sycophant is a calendar application which generates appointment reminders differently according to user context, for example as a visual note or as speech; the user is then asked to indicate whether system behavior was appropriate. (Schmidt-Belz et al., 2003) in turn describe the CRUMPET system, a personalized, location-aware mobile tourism service, and ask whether users "feel the service has added benefits, compared to traditional media and Web-based services". The system recommends tourist attractions based on personal interests and vicinity to the current location, on a handheld computer. In evaluation, over 75% of users considered the system to possess added value compared to "existing media", meaning printed materials. The intrinsic added-value of adaptivity within the system itself is not addressed. In addition, a usage scenario for tourists can be considered particular in the sense that (tourist) users are people for whom it can be assumed they are interested in novel features, such as pro-active tips regarding what do visit next.

A general problem in measuring how context-awareness affects usability is the lack of an appropriate comparable basis: a context-aware research prototype fulfilling a certain function can usually not be compared with a context-unaware counterpart fulfilling a similar function. In the estimation of (Schmidt-Belz et al., 2003), "there is certainly a trade-off between avoiding cognitive overload, giving task-specific information, and adapting to personal preferences, which needs further and more detailed investigation."

Chapter 3

Context in Web Applications

This chapter discusses how context-aware computing can be used in Web-based systems and introduces a modeling approach to Web Engineering taking context into account. We start by discussing possible effects of adaptation to context that can occur in a Web application. Subsequently, we informally describe context with scenarios for context usage, before proceeding to a formal definition of context and showing, at the conceptual level, how it can be used in Web Engineering.

3.1 Adaptation to Context

Context-awareness can be a basis for adaptation in a Web application. If a system is aware of information about context of usage (see Section 2.1), the system can adapt to the user's presumed requirements, specifically in situations where informational constraints or physical constraints are high.

Adaptation can occur in various aspects of an application. To better analyze and support adaptation effects, we distinguish three levels in a Web application: navigation, content and presentation. Potential adaptation effects for each level are summarized in Table 3.1.

Level	*Adaptation*
navigation	(i) show or hide parts of the navigation structure, (ii) dynamically (re-)construct the navigation structure, (iii) offer additional links.
content	(i) select relevant resources (documents and services), rank them, (ii) adapt the documents themselves, by selecting parts and integrating them into a final document, (iii) compute parameters of integrated services according to context.
presentation	(i) adapt layout and other user interface characteristics, (ii) adapt to (potentially changing) constraints of the user's physical device.

Table 3.1: Adaptation effects in a Web application

The navigation level describes navigation possibilities and how these are structured. Adaptation can affect which possibilities exist and alter their structuring. An element of navigation in a classic Web application is presented as a link that can be followed upon a mouse click; however, how navigation possibilities are in fact graphically presented to the user is not a concern addressed at the navigation level.

The content level describes the actual Web application offering. This can consist of information presented to the user, and of operational features that are available to the user. We call "service" the mechanism through which the application provides the user with an operational feature: for example, a reservation service through which the user can book a flight. Typically, the Web application presents such a service to the user, and to implement the service relies on an actual, back-end service, that is either locally available or is a third-party offering. The Web application thus acts as consumer of this service. Generally speaking, adaptation can influence which elements are offered, and the way in which an element is determined.

The presentation level is responsible for the appearance of the user interface to the application (the GUI). Adaptation of appearance has been particularly stressed in ubiquitous computing: for example, in (Schmidt, 2000) the user's current physical situation such as "the user is moving" is used to increase font-size to ease reading.

Enhancing system usability is a major goal of adaptation. In this work, using context in a Web-based system means to use context to enhance the user interface in the largest sense. Interface look and feel, but also content and services provided in a certain situation are selected according to context, and may themselves be adapted to context.

Some approaches considering context in applications are of a general nature, for example conceptual considerations for arbitrary software systems (Brézillon, 1999). We argue that, for engineering purposes, Web-based systems require a specific focus regarding context, to address the particular characteristics of these systems (see Section 1.1). For the engineering process, a comprehensive, yet comprehensible context model is needed: the model must be powerful enough to enable meaningful adaptation, yet sufficiently straightforward to be used by the people involved in the process. These goals are difficult to amalgamate, and thus any approach attempting to address both goals involves a compromise between power of expression and simplicity of understanding.

3.2 Context Usages and Characteristics

The notion of context-aware computing was initially used to adapt applications in mobile usages according to location of usage. As illustrated by the examples in Section 2.1.1, current research increasingly demonstrates the potential of context-awareness in a wider variety of scenarios. Our work contends that context-awareness can be useful for Web-based system usage in the large, if extended to further context aspects. Context-awareness can then be used for adaptation in systems in mobile and non-mobile usages; whereas mobile usages in particular can benefit when additional aspects to location are considered.

Use of context-aware computing in applications can be categorized in several ways:

- according to what types of context are being considered, such as location or time;
- according to application domain, such as electronic tourist guides or health services;
- according to type of technology employed: Web technology, embedded devices, augmented reality devices.

Such categorizations can be considered complementary views, the view in question being useful to a certain type of viewer: the type of context used might be most important to an application modeler and designer, the application domain to a financial sponsor, and the type of technology to engineers and developers. Of special interest in our work are the types of context that can be of influence, their characteristics, and how this information can be taken into account in a systematic way.

3.2.1 Example Scenarios

We now introduce two scenarios for Web software in which context-awareness can be used, and which motivate the types of context used in this work.

Context in e-Commerce

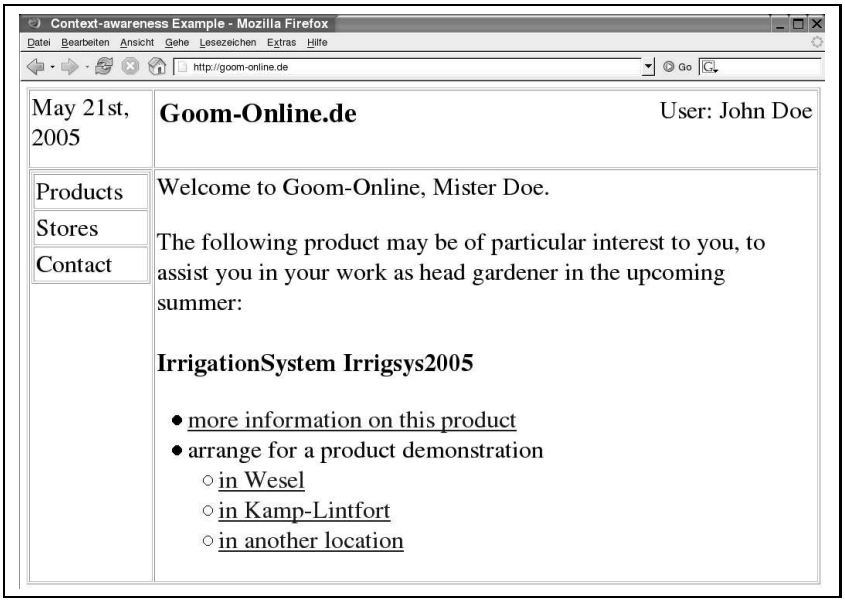

Figure 3.1: A context-aware e-commerce Web application

Consider a Web application for a construction market servicing various geographical areas. This application offers a Web presence to the company's actual outlets, and possibly an online store. Figure 3.1 shows a scenario in which the user is known to the system and the user's profile is used in order to predict which types of products might be of interest to this user. Furthermore, the date is used in order to provide season-relevant information. The application shows the user a product which is potentially of interest, and gives the user the ability to gain further information on this product, via a product description document attached to this page. In addition, the application offers to arrange for a product demonstration in one of the store locations that the system assumes are near the user, however does not preclude other possible locations. Location can thus also be useful in this scenario. This type of scenario, selection of products according to context, is analogous to features offered by recommender systems (Schafer et al., 2001; Burke, 2002).

Generally speaking, what is presented to the user are context-relevant pieces of information, but also further possibilities for interaction with the software: we term these *services*. Such a service is thus also offered in a context-aware manner, and service incorporation into the application is also context-aware in that context information can be used to parameterize a service. To summarize: in a context-aware Web application, navigation, content selection, service offering, and display type may be adapted according to context.

Industry Maintenance Scenario

Consider a technician responsible for maintaining industrial installations. The technician uses the company's Web-based system in various situations. When in the office, an Intranet portal is accessed via a desktop browser to gather knowledge regarding company products or to update work accountancy. When away from the office, the technician uses a PDA[1] to check and update the appointment list. When an installation is visited, the installed product is scanned with an RFID[2]-reader integrated in the PDA to read the product identification. The device then retrieves details about this product from the company's database, via a Web Service, assuming the technician has network access at this site, for example, via a wireless Internet connection. If furthermore the system knows (or assumes) the user's current task is to perform a maintenance inspection, the application can provide in addition a graphical user interface for regulating the inspected product via a network interface. This user interface is generated taking into account information about the current situation of usage, such as the technician's profile. Once the technician has returned to the office, if the PDA is used, the system might propose a synchronization service of any work steps performed offline on the PDA.

In this scenario, the system uses various sorts of context information to adapt the offering, such as user role (technician) or the hardware device used (desktop, mobile device). Location information can also be used, although coarse-grained in this scenario: in the office, or away from the office. The current task (such as make a maintenance inspection or check for appointments) can likewise be used as input to adapt the system, and may thus also be considered context. This is a different perspective on context than in (Dourish, 2004), where "context arises from the activity", and the activity or task is in itself not classified as context. Time information can be used to further fine-tune the offering: if current time is within usual maintenance hours, it is more likely the user is making an inspection call, and so this option might be proposed first. If not, a different option such as "update appointment schedule" might be presented first.

If the system is aware of these aspects of context, it can adapt to provide the user with the most likely features in a given situation; furthermore how a feature is presented can also be steered by context of usage. The system should not disable other possible features, as the precise context of usage can not be known for certain.

An additional point of interest is that context information can be used to extrapolate supplementary information about context itself, thus potentially permitting a higher degree of context-awareness in the system. In the example scenario, if the device used is a mobile device, the user is outside of the office and time is within routine maintenance hours, the system might deduce that the user's current task is "make an inspection call". As stated above, we consider that this information about the task *is* context, as in (Dey et al., 2001): "several pieces of context information that describe a situation [are used] to infer a new piece of context information". The system however can not be certain of this deduction, and thus various strategies are conceivable for deciding when to take steps based on context information. In some situations, an explicit user confirmation may be required.

Further scenarios for context-awareness

As discussed in Section 2.1.1, other research projects have described scenarios of a different nature for which the notion of context-awareness is applicable, typically tourist guide scenarios or ubiquitous computing situations aiming at supporting an office environment. It can be assumed that additional types of scenarios will be described in the future. For instance, portals and news sites might use con-

[1] Personal Digital Assistant
[2] Radio Frequency Identification

text information to provide customized, tailored views (in addition to existing, explicit personalization features). The context paradigm might, in the long term, permeate the underlying software systems themselves: the K Desktop Environment (KDE) for example is projecting to integrate a "Contextual Linkage Engine" in the desktop (KDE Community, n.d.), whose goal is store context information about documents and their usage on the personal computer.

3.2.2 Categorization of Context Elements

We now address the question of how context information can be organized. It must be determined which factors of context are relevant in an application domain, and how these factors are then related to application models. To address these questions, we begin by discussing a categorization of factors of context in this section. The following section (3.3) describes how this information can be used to define a context model.

In an engineering process, specifically one aiming to take context into account throughout the process, methods and models that explicitly address context are required. The question is then which sort of context information can be explicitly represented, and to what degree. This *context base* is established during the Web Engineering process, and a run-time system then determines *context knowledge* for a specific situation of usage, according to this context base.

The manner in which context is determined during system operation varies significantly between approaches, and can be achieved directly by the system providing the Web application or indirectly by delegating to extraction techniques. In any case, some form of explicit representation of context is necessary, upon which these techniques can then be applied.

Factors of context relevant to an application domain must thus be identified and explicitly accounted for. Individual factors can have commonalities or relations between them. For example, "summer" is an instance of a "season", which in turn is a sub-category of "time". To represent this sort of knowledge, we propose to use ontologies (see Section 1.1.3). This structuring allows to use relations between context factors themselves to gain further knowledge about context, without requiring that all context ontology items be explicitly put in relation to application elements. Furthermore, the use of ontology mechanisms to represent context factors enables sharing context catalogs. Such catalogs can be defined independently of a specific application, for example by a standardization body (assuming this body uses ontologies to represent such factors).

We furthermore propose to use one ontology for each type of context, that is, to assign each factor relevant for context to a specific category. The actual categorization that is chosen places no restrictions on the context model described in this work, nor on implementation, and can readily be replaced by alternative categorizations. The general purpose of subdividing the set of context factors into categories is to facilitate the modeling process, and the implementation of corresponding tools as well as a run-time system for adaptive Web applications. Regarding modeling, it is assumed that some form of human interaction is required to complete and adjust the model, although automatization techniques for context models can be developed to assist in this task.

The grouping of factors is then indispensable for the user to maintain an overview of the model. In particular, domain experts, assisting in the Web Engineering process, need a manageable view of the context model. In our approach, modeling context with regard to the application consists of associating factors of context with application domain elements (details are given in Section 3.3); for this activity as well, categorizations makes the model more manageable. Furthermore, the categorizations can be used to refine the way in which context information is to be used in a specific situation; for example, the designer can then specify that, for a certain adaptation feature, only a subset of the categories of context should be considered.

Thus, context categories that, on the one hand, make the modeling process manageable and on the other hand, cover as many scenarios for context as possible are helpful. Given that we wish to classify context elements, we now should ask which, and how many, classes are useful. Existing approaches in mobile scenarios, as described above, usually focus on one or two classes, that are key to the scenarios they cover: these are typically *Location* or *Device*, sometimes *Time*. The limitation to one or two classes has the advantage that, in theory, the model is easier to understand; however, to support a wide variety of scenarios, additional sorts of context factors must be taken into account.

The question is then what categories are most effective at representing a broad variety of Web applications, while remaining easy to understand by modeling users. Ideally, categorizations will be established over time as "best practice". This dissertation uses a concrete set of categories, in order to provide tangible application models and examples. However, which categories are ultimately used is not a critical issue for the approach discussed in this work.

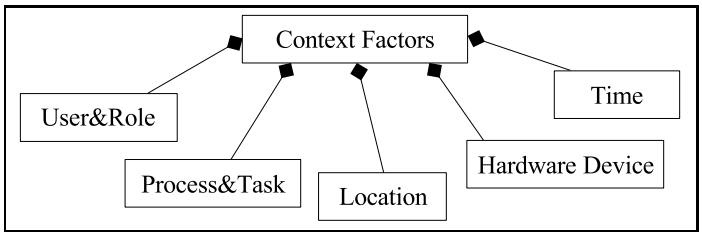

Figure 3.2: Context factor categories

We propose the following categories of context factors (see also Figure 3.2), as motivated by the example scenarios of Section 3.2.1: *User&Role*, *Process&Task*, *Location*, *Time*, *Hardware Device*.

- *User&Role* provides a categorization of users according to their role, such as various types of customers, or different types of employees. This can include context information gathered through observing behavior, for example, to determine a personality type. An individual user may have several roles.

- *Process&Task* represents a functional context, such as work items for employees or an online-shopping process. A process may be modeled, for example, as a collection of tasks belonging to this process.

- *Location* is a categorization of locations relevant to the application, in the desired granularity: for some applications, country might be sufficient location information, for others, city, and so forth; in other cases the distinction between intranet and external access might suffice. These locations are not to be confused with physical locations *sensed* by the system, such as by GPS positioning, user input, or network address (Internet address or the like). Furthermore, this location categorization does not refer to locations in the information space (meaning, where is a resource located) nor their proximity location-wise to other elements in this space: in the terminology of this work, this type of relationship is part of the domain ontology, not of the context categories.

- *Time* refers to various types of time information that can be relevant, such as a client's time-zone, actual time, or a virtual time.

- The *Hardware Device* category contains that device and device-related information which can be relevant context factors. This can include device type (desktop, mobile) as illustrated in Section 3.2.1; display properties affecting look and feel, but also physical properties of the system

environment: for example, a property such as bandwidth can be used for (automatic) application adaptation. If bandwidth (as a context factor) is sensed to be below a given threshold, or changes to fall below that threshold, streaming of audio or video can be reduced in quality, in order to maintain the same refresh rate nonetheless (Nepal and Srinivasan, 2003).

Other approaches at Categorization

Related work aiming at integrated modeling of context factors sometimes makes more abstract categorizations, such as "physical environment, social environment" (Schmidt et al., 1999), or "material context, social context" (Petrelli et al., 2001). Such categorizations can be useful for high-level reasoning about context and possibly for experts in knowledge representation. It is our belief that the categorization proposed above follows a more user-centered terminology. The goal of pursuing a user-centered terminology is to facilitate the modeling process for modelers of Web applications, and, consequently, to enable emergence of meaningful models. Which categorization is most effective for this purpose can only be ascertained by experience in engineering projects, and furthermore likely depends on the application domain. For example, in an architecture such as CAMUS (Ngo et al., 2004) focusing primarily on ubiquitous computing (as a "web of devices"), it can be argued that a further category *environment* is useful, representing factors such as air humidity or light intensity.

Another frequent approach at categorizing context factors is system-based, meaning that the model is structured according to those categories of context that are useful to the context system; see, for example, the AmbieSense project in (Kofod-Petersen and Aamodt, 2003). For the reasons described above, this dissertation describes a more user-oriented model; this does not preclude transformation into a system-oriented model for the runtime context system.

Context information can also consist of user behavior as gathered over time: which links did the user follow previously, what search terms were used, and so forth. (Lieberman, 1995) describes the Letizia system that tracks user behavior in order to anticipate items of interest; upon request, a page containing current recommendations is presented: a feature understood as adaptation in our work. In our terminology, navigation within the application results in activation of resources in the application domain, that is, in determination of a domain context. Inferences regarding current type of user instead result in activation of context factors in the category *User&Role*.

3.2.3 Aspects of Context

Context has various characteristics. Some characteristics are related to the nature of context in itself, whereas others only appear in combination with a system. We thus distinguish *inherent context characteristics* and *system augmented context characteristics*, see Table 3.2.

inherent context characteristics			system augmented context characteristics		
regular	⇔	irregular	synchronic	⇔	diachronic
low-frequent	⇔	high-frequent	explicit	⇔	implicit

Table 3.2: Context characteristics

Inherent Context Characteristics

Two dimensions of context characteristics are inherent, in the sense that they are located directly in the context factors: regularity and frequency. These dimensions are best illustrated by analyzing how context information, as categorized in Section 3.2.2, relates to these characteristics.

First, context factors can be distinguished with regard to regularity with which they change. *Time* itself changes regularly, independently of how it is used in the context model. *Location* information on the other hand does not change regularly: the user moves and stops in an unpredictable fashion. Likewise, *Process&Task* is an irregular context category: in the example scenario of Section 3.2.1, it is the user who chooses to engage in a process (even though the user can be assisted by the context-aware system in process selection). Change in process is thus unpredictable and irregular, although in some scenarios, a process, or subdivisions of a process, for example specific tasks, might be activated by a system at pre-defined intervals. *Hardware Device* and *User&Role* context information are likewise irregular.

The other dimension describes the frequency with which the context changes. This frequency depends on a specific view on context information. *Time* information has a high change frequency if, for example, seconds are considered; however if season changes are considered, frequency is much lower. *Location* is similar, as the user's position can be viewed in different levels of accuracy, such as country or city. Change in *Process&Task* depends on whether a global process, such as maintenance of a machine in the example scenario, is considered, or whether smaller dependent subtasks are considered. *Hardware Device* and *User&Role* information change slowly or quickly depending on the scenario.

Both regularity and change frequency of context are not independent of specific views, yet are inherent characteristics of context. The Web engineer must thus decide in what intervals the system has to update its knowledge about context, notably when external sensors are involved.

System-augmented Context Characteristics

Two further dimensions of context characteristics are not context-inherent, but only appear when context is used within a system. These dimensions describe the level in which the system augments context information. For that reason, we call them system-augmented context characteristics.

One aspect is the level to which historical context information is taken into consideration. Previous context information can augment the currently given context. If the system knows the technician's previous actions in the example scenario of Section 3.2.1, the system can use this information to support the user adequately in new situations, for example, by remembering a difficult maintenance step. In such a case, the system uses what we call the *diachronic* character of context. On the opposite, a system taking only current context into consideration, without referring to previous context, uses context *synchronically*.

A second aspect is the degree of interpretation of context information. Some context information requires little or no interpretation to be gathered from input data, for example, a location. Other context information can require inference mechanisms in order to be sufficiently determined: in the industry maintenance scenario of Section 3.2.1, the process (as a context factor) can be inferred from other context factors. We thus distinguish between *implicit* and *explicit* information, as in (Siljee et al., 2004).

Application of these aspects depends on available system mechanisms. If the system does not store previous context information in any way, the diachronic character of context can not be accounted for. Likewise, if the system does not have inference mechanisms, implicit context data can not be used.

Further system aspects also influence the level of context-awareness, such as quality of the context retrieval process, and whether the used context model is static or changeable. We call these the context-related system characteristics. Within the Web Engineering process, decisions must be made regarding which of these characteristics are significant for a specific application, and thus justify potentially higher development and production costs.

An alternative classification of these aspects of context is mentioned in (Dourish, 2004), who distinguishes between "the positivist and the phenomenological positions" with regard to context. Context factors such as device information are stable, in the sense that they are features of the world, independent of actions of individuals. A phenomenological perspective takes into account activities, or, in the terminology of this work, *Process&Task*.

3.3 A Unifying Context Model

In order to support adaptation according to context, knowledge about this context must be defined and represented in a suitable manner. A conceptual model, based on a precise definition of context, enables to define known context information. Modeled knowledge forms a basis for adaptation in the Web-based system; such knowledge is however not exhaustive, as information gathered about the user, including patterns of use can ultimately also constitute context information.

As stated in Section 3.2, this work aims to provide for context-awareness for a wide variety of scenarios. Consequently, an approach to modeling context on a broad base is presented, where context is integrated with application elements. In this modeling approach, the domain ontology plays a central role and might already have been developed in the overall engineering process. The goal is then to connect the domain ontology with context information. Context factors are classified and put in relation with domain ontology elements. This approach is detailed in this section; the subsequent section (3.4) focuses on how this model can be used in Web Engineering.

3.3.1 A Formal Definition of Context

The general notion of context in this work is that elements of the conceptual design space of an application, such as concepts defined in the domain ontology, can be contextualized through relations to elements of the context space. An information object can be, for example, relevant for a geographic region or location. This relation might be reciprocal: in the example, the location might be relevant if current context includes the focus on that information object.

We therefore say that context is determined by:

- a structuring of factors pertaining to context into ontologies;
- an ontology describing the information of the application domain;
- relations between the ontologies, indicating which elements are context-relevant with respect to which other elements;
- a current perspective on the ontologies. This perspective consists of a subset of each ontology involved and a degree of activation of each involved ontology entry. This can be termed contextual knowledge, as in (Brézillon, 2004).

These notions of context and context relations are formalized in the following definitions.

As our context-related definitions are based on the use of ontologies, a definition of ontology is first required; we follow the definition as given in (Wissen, 2004b):

Definition 1. *An* ontology *is an 8-tuple* $O = (C, R, L, F, G, H^C, rel, A^O)$ *where*

- C *is a set of concepts and* R *a set of relations;*
- L *is a lexicon consisting of a set* L^C *of symbols for concepts and a set* L^R *of symbols for relations, such that* $L = L^C \cup L^R$*;*
- $F \subseteq L^C \times C$ *and* $G \subseteq L^R \times R$ *are two relations representing references on concepts and relations. For* $l \in L^C$ *let* $F(l) = \{c \in C | (l, c) \in F\}$ *and* $F^{-1}(c) = \{l \in L^C | (l, c) \in F\}$*; for* $l \in L^R$ *let* $G(l) = \{r \in R | (l, r) \in G\}$ *and* $G^{-1}(r) = \{l \in L^R | (l, r) \in G\}$*;*
- H^C *is a concept hierarchy, i.e., a directed relation* $H^C \subseteq C \times C$*.* $H^C(c_1, c_2)$ *means that* c_1 *is a subconcept of* c_2*;*
- rel *is a function* $rel : R \rightarrow C \times C$ *that describes non-taxonomic relations between concepts. The function* $dom : R \rightarrow C$ *where* $dom(R) = \Pi(rel(R))$ *defines the domain of* R*, the function* $range(R) = \Pi(rel(R))$ *defines the range of* R*;*
- A^O *is a set of axioms, i.e., expressions in a logic description language.*

We now describe how elements pertaining to context can be defined, and put in relation to other elements to define context relations.

Definition 2. *A* context domain C_d *is defined as a union of ontologies:*

$$C_d = O_1 \cup O_2 \cup \cdots \cup O_n, \quad O_i \text{ is an ontology}$$

Definition 3. *The* degree of activation a *of an element* e *of a context domain* C_d *expresses the certainty with which the element is known to be active:*

$$a : C_d \rightarrow [-1, 1], \quad C_d \text{ is a context domain}$$

If an ontology element is chosen by the user in the run-time Web application (for example, when the user chooses a navigation option generated from this ontology element), the degree of activation of the element is 1, whereas if an element is inferred to be active by some system mechanism, the degree of activation is set by the inference mechanism.

A context relation describes that two or more elements of C_d influence each other's degree of activation, meaning that when one or more of these elements is/are active, the other element(s) consequently has/have a certain degree of relevance. A context relation is represented as a set of pairs of elements from a context domain C_d. This set contains only one pair if the relation involves precisely two elements from C_d. The set contains several pairs in case of an n-ary relation. The set of knowledge about context relations that can be expressed is thus the set of all such sets of pairs (i.e., the power set $\mathcal{P}$). We call this the *context space*:

Definition 4. *The* context space C_s *on a context domain* C_d *is defined as:*

$$C_s(C_d) = \mathcal{P}(C_d \times C_d)$$

Definition 5. *A* context relation c_r *is a point in a context space* C_s*; that is,* $c_r \in C_s$*.*

To illustrate the representation of an n-ary relation where $n > 2$, we consider a ternary context relation. Such a relation can be represented as a set of two pairs of binary relations. For example, let c_1 be a context relation between factors d_1, p_1 and p_2; this can be written as $c_1 = \{(d_1, p_1), (d_1, p_2)\}$.

Using sets of pairs to represent context furthermore allows for ordering. Each pair in a context relation space is implicitly ordered; however whether this ordering is significant is left open for a specific context model interpretation to specify. A model in which ordering is in principle relevant, and which needs to express that in an individual relationship both directions are equally relevant, writes, for example, $c_2 = \{(d_1, p_1), (p_1, d_1)\}$.

A context relation possesses a value which is computed by an *influence function*: function:

Definition 6. *An* influence function i *computes the influence of a context relation* c_r*, as a real number between* -1 *and* 1*:*

$$i : C_s \rightarrow [-1, 1], \quad C_s \text{ is a context space}$$

Intuitively, a context relation between two items indicates to what degree the items are relevant together: when one item is active, the other is relevant to a degree determined by this relationship. For example, if $\{(d_1, p_1)\}$ is a context relation and d_1 is active, the relevance of p_1 is determined by i and the degree of activation of d_1.

Definition 7. *The* contextual knowledge C_k *is the knowledge a system has at a specific point in time during its operation about which ontological elements are active, and to what degree:*

$$C_k = \{(e, a(e)) | e \in C_d\}, \quad a : C_d \rightarrow [-1, 1]$$

The contextual knowledge is knowledge which results from the context model and a system's context inference mechanisms. It is as such not part of the context model, but instead expresses the knowledge resulting from the context model at a specific point in time within a system. Its computation is not predetermined by the context model, but is dependent on specific strategies, which may use the influence function, the ontological relations, and any combination of these.

3.3.2 Concrete Context Model

We now introduce a concrete interpretation of the context model of Section 3.3.1; this interpretation is designed for supporting context modeling for Web applications in a flexible manner, while nonetheless being of manageable complexity in view of modeling processes. This context model is used throughout this work for the illustrations of context-awareness and adaptation in example scenarios and system architecture.

While automation techniques for context model generation can be developed to assist in creation of such a model (or parts of it), it is indispensable for an expert user to be able to validate and extend the model. Therefore the model needs to be at the same time flexible with regard to representing context information, and comprehensible by an expert user. The goal of attaining this compromise motivates the design decisions presented here.

Definition 8. *Given the categorization of context factors introduced in Section 3.2.2, we define the* context domain for Web Engineering C_{dw} *as using the following ontologies:*

$$C_{dw} = U \cup P \cup L \cup T \cup H \cup D$$

where U is the ontology of user and role factors, P the processes and tasks, L the locations, T the time factors, H the hardware device factors and D the domain ontology.

In our concrete context model, the influence function is a weighted association between ontology entries, and involves two or more entries, of arbitrary types. Stated more precisely, influence is defined as follows: an influence value function i_w associates a fixed value to a context relation c_r. We call this value the *influence value*. In a binary relation, when one element is active, the other is considered relevant according to the influence value and the degree of activation of the first element. In an n-ary relation, the relevance of an element is determined by the influence value and the degree of activation of the other elements in the relation.

How the information in the context model is used for computation at run-time is not restricted by this model; the model may therefore be used by varying strategies and algorithms for context inference. Usable information consists, on the one hand, of the context relations defined as above, and, on the other hand, of any ontological relations involving elements of the context relations. A concrete strategy must define, notably, whether influences are cumulative, and whether context-relevances determined via influence values should propagate to further ontology elements. Possible effects of ontological relations are discussed in the following.

3.3.3 Context Knowledge and Relations

Given that context factors are associated with elements of domain knowledge, this raises the question of what contextual knowledge can be gathered on items which are not in this association but are in an (ontological) relationship with items which are explicitly linked in a context relation.

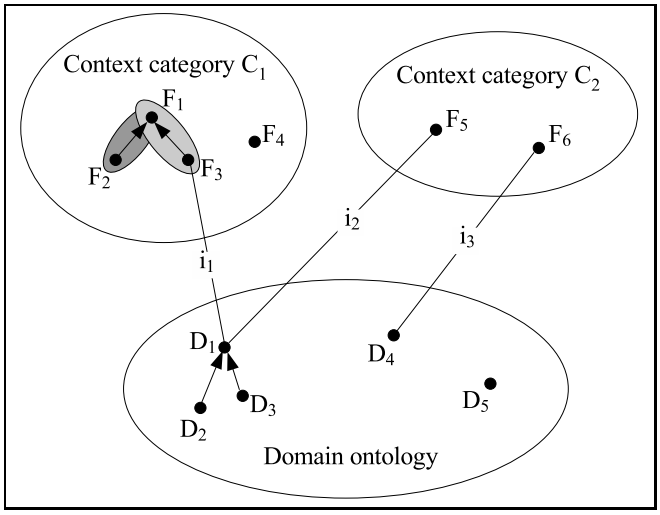

Figure 3.3: Context knowledge and relations

We therefore consider the effects of ontological relations on contextual knowledge. An illustration of possible effects is shown in Figure 3.3. In a specific context, certain context factors are active, with different degrees of activity: here, presume context factor F_3, belonging to category C_1, is sensed as being active with a value of 1. The context model states that domain ontology element D_1 is context-relevant as D_1 is related to context factor F_3; the relevance value is determined by the influence value i_1 and the degree of activation of this context factor.

In addition to explicitly linked elements, contextual relevance can be asserted for further elements according to the type of relations within the domain ontology. Elements D_2 and D_3, being in relationship with D_1 might also be relevant; however whether this relationship is significant and to what degree it transposes relevance from D_1 to D_2 and D_3 can not be said in a general manner. Concrete

adaptation specifications need to define which sorts of relationship are to be followed in which situation, and how these relationships affect relevance; for example, there might be a relevance loss on following a relation in this manner.

Default strategies and interpretations can be postulated. In the example of Figure 3.3, relations represented by an arrow signify a hierarchy. In such a hierarchy, when a higher-level element such as D_1 has relevance, a default interpretation can be that D_2 and D_3 being sub-items of D_1 are also relevant, with no loss of relevance with respect to relevance of D_1. Additionally, a default interpretation can consider that context relations on domain elements that are classes also transfer to elements that are instances of that class. The converse however need not apply; relationships from a child to a parent or from an item to a sibling can not be used as a general statement to infer relevance, although such relationships might be relevant in individual situations. Now suppose context factor F_5 from category C_2 is also active. This factor is also connected to D_1, which is already relevant by its connection with F_3. In this case, a default interpretation for the relevance of D_1 can be that the maximum influence value between i_1 and i_2 prevails. Alternatively, a strategy can postulate that influence values augment each other in some way, in this case increasing the overall relevance of D_1.

Knowledge about the domain can also be used to infer knowledge about context. For example, suppose a user action, via application navigation, results in activation of domain element D_4. The model states that context factor F_6 can be considered relevant with a value factored by i_3. Thus, given a resource, the model can provide context factors for which this resource is considered relevant. This knowledge might then be used for further context deduction; or for directly assisting the user, for example for semantic retrieval (Varelas et al., 2005), to search for further resources relevant in that context.

In summary, contextual knowledge in our approach is based on two sources of knowledge representation. Ontologies represent a *semantic network*, that is, a network between elements where the edges are labeled. These ontologies represent knowledge about the application domain and about appropriate context factors. The context relations in turn form an *associative network*, that is, a network where the edges have a numeric weight. The combination of these two sources of knowledge can be used to compute contextual knowledge in a situation of usage.

3.3.4 Example Context Models

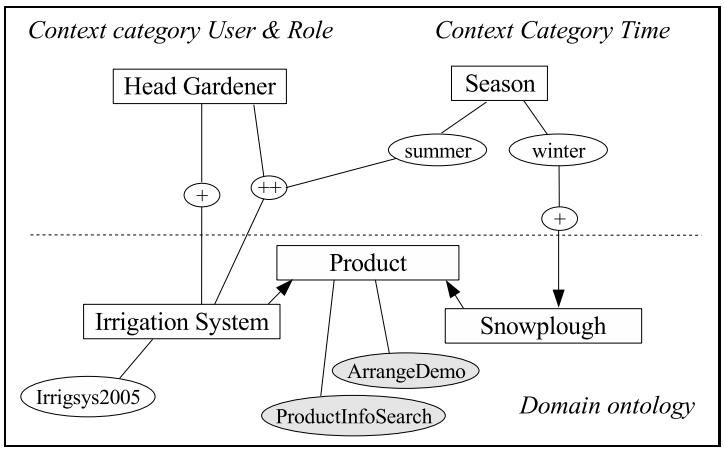

Figure 3.4: Context model for an e-commerce scenario

Figure 3.4 shows a context model reflecting the example scenario discussed in Section 3.2.1. In

the diagram, a class is represented by a rectangle, an instance of a class by an oval. Arrows between classes represent a specialization relationship (subclass). Services are additionally emphasized via a grey background. A context relation is represented as a connection between two or more items; when this relation is ordered, an arrow represents the ordering. Context relations are furthermore is labeled with a weight; these weights are expressed as a discrete set of symbols, such as $+$ and $++$. Although the definition of context allows for arbitrary influence values, it is presumed that a usage of discrete values yields more meaningful results, given that influence values need to be entered (or at least verified) by a business expert in an intuitive manner.

In this model, a type of product from the domain ontology ("Irrigation System") is said to be relevant (marked as $+$) for a certain user profile ("Head Gardener"). Furthermore, in a certain season ("Summer"), and for the same user profile, the product is modeled to be even more relevant (marked as $++$), making the element more susceptible in appearing, for example, in a customized offer. All product instances (such as "Irrigsys2004") attached to this product type can then be used in that context. Attached to "Product" are two services, "ArrangeDemo" and "ProductInfoSearch", that may be offered to the user for any product. The modeler has further indicated that in the season "winter", the entry "Snowplough", a specialization of product, is relevant, and has explicitly stated that this context relation is ordered from the former to the latter.

As shown in this example, the business user models context relations where deemed most appropriate; not all elements need be explicitly linked in such a relation. As discussed in Section 3.3.3, influence values may be determined for items not explicitly in a context relation: in this example, the product "Irrigsys2004", as in instance of an "Irrigation System", might be considered context-relevant when the class "Irrigation System" is determined to be context-relevant. Whether this is the case, and to what degree, depends on a specific strategy.

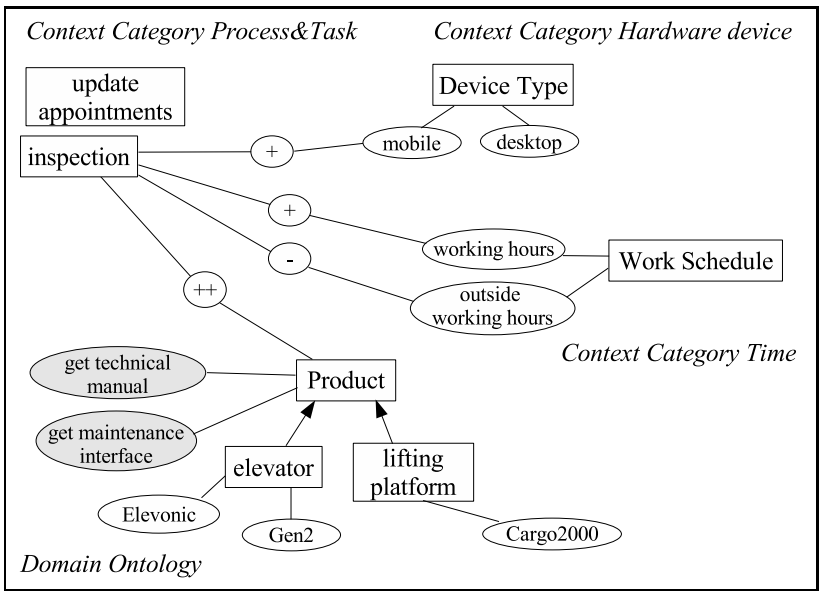

Figure 3.5: Context model for an industry maintenance scenario

Figure 3.5 shows a further example context diagram; this diagram is a possible representation for a context model supporting the industry maintenance scenario which was described in Section 3.2.1. Types of product (e.g., "elevator") are domain knowledge, as are instances of products ("Elevonic"). Services such as "get technical manual" are associated. The model states that, if the task "inspection"

is active, then the concept "Product" is deemed highly relevant (value $++$); the opposite is also true. Furthermore, if the context hardware device parameter is identified as "mobile", the task "inspection" is somewhat relevant (value $+$). If time is recognized as being within working hours, the same task is relevant. If both factors are given, the relevance of the task "inspection" is more likely.

The diagrams in Figure 3.4 and Figure 3.5 show one possible view of context information to be represented in a system. Not all modeled knowledge can be shown in this fashion; a model may need to support more complex rules as well. The current view on context information should ideally depend on the level of expertise of the user involved in the modeling process. How knowledge is then represented within a system is independent of graphical representations intended for users.

Influence Values

The issue of precision of an influence setting was introduced by the example in Figure 3.4. In principle, it is necessary to maintain the possibility of arbitrary precision: an influence value need not be explicitly set by the user, but could either be the result of automated extraction, or be a result of an item's relation to other items that are involved in explicit context relations. For the latter, an algorithm computes the actual relevance value at run-time. For such items for which the modeling user wants to set an influence value, the possibility of entering a high precision value (such as a probability value) is presumed to be counter-productive, in the sense that it may result in confusion and errors. Instead, the business expert might specify an influence value as an element from a small set of discrete values, for example: "$++$" meaning highly relevant, "$+$" meaning rather relevant, no marking meaning somewhat relevant, "$-$" meaning rather irrelevant, "$--$" meaning quite irrelevant.

A "$-$" or "$--$" setting indicates items that are irrelevant or inappropriate in a given context, and thus can bother an end-user, or even be considered offensive if presented in a certain context. Consider the following example to illustrate the need for such a relation: a retail store services many types of consumers, and products based on a certain type of meat should not be recommended to customers of whom the system knows they are of a religion that forbids consumption of such a product. Thus, this knowledge could be modeled by a "$--$" relation. Note however that one should not rule out entirely offering this product for this type of customer: the system will make an assumption that this product is likely irrelevant for this user, however there are situations where it is not, for instance if this particular user is not concerned with such constraints, or is shopping for somebody else who is not concerned. Alternatively, the latter scenario could also be modeled as a task "buy for a friend" within the *Process&Task* context category, in which case the user role factor would not be used in the same manner. To summarize, the system should avoid recommending a product if it is probably irrelevant, however the system should not forbid showing the product altogether, for instance consequently to the user explicitly choosing it, by navigation or searching.

3.4 Context Modeling in the Web Engineering Process

This dissertation proposes an approach to taking context into account that augments the WISE methodology (see Sections 1.3 and 2.3).

An overview of the modeling process used in our approach is given in Figure 3.6. As recommended by WISE, four separate sub-models describe the application model as a whole, where each sub-model can refer to information defined by a proceeding model (order of precedence is indicated by grey arrows). Once established, all model information is stored in a model repository (as indicated by the white arrow).

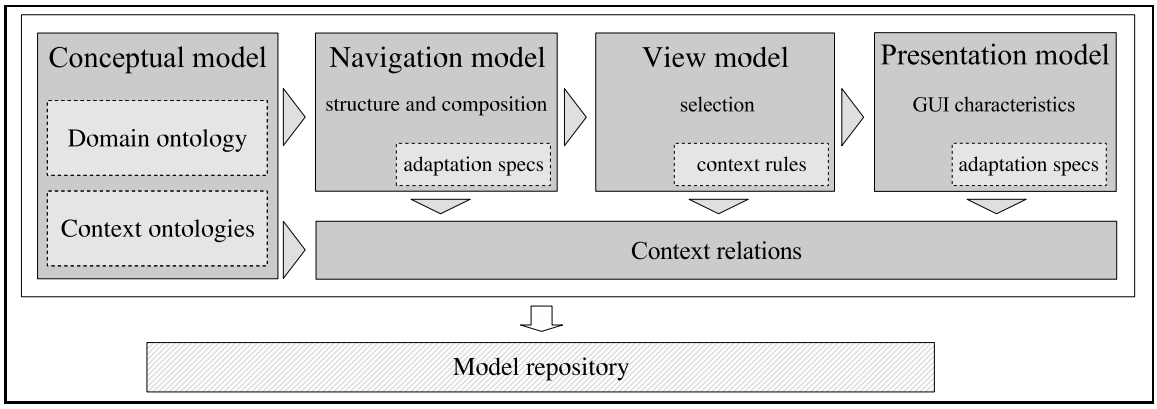

Figure 3.6: Context modeling augmenting the WISE methodology

We now provide an introduction to the models and the relations between them; the individual models are then detailed in Section 3.5. The conceptual model describes the application domain, including content building blocks (or references to location of such items) that are to be presented to a user. The conceptual model is extended by ontologies of context factors, one for each category of context to be used (see Section 3.3.2). The navigation model defines navigation possibilities for a user by referring to elements in the domain ontology. Furthermore, the navigation model defines how to compose a Web page when the user has selected a navigation item. The model is extended to allow for adaptation, for navigation as well as for composition. Placeholders for adaptation can be used to refer to an adaptation specification whose properties define the strategy for that placeholder. The view model defines which elements of the navigation model are to be viewable by the user in which situation. The presentation model defines GUI characteristics.

Context relations can now be defined on the models, as weighted associations between items in these models. A relation between the domain ontology and a context ontology indicates that a domain element is specifically relevant when a certain context factor is active. Such an association with the domain ontology can be with any sort of domain element, such as a concept in the ontology, an information item attached to a concept, or to a service available to the Web application (or, more precisely, a service description). Context relations can likewise be defined on elements of the other application models. For example, a class of GUI characteristics (in the presentation model) can be associated to a user role, indicating that this sort of presentation is more relevant for a certain type of user.

Throughout the context modeling process, it should be kept in mind that the scope of context modeling is that scope which is relevant to the immediate requirements of the Web application being modeled, and does not include knowledge of the internal operation of external, integrated systems (see also Section 1.1.3).

Modeling Scenario

A typical modeling scenario in the methodology (as augmented for context) might be as follows:

1. the application models are established independently of context-awareness considerations,

2. a business expert extends the conceptual model with categories of context and subsets of these categories appropriate for the application,

3. items from the domain ontology are put in relation with context factors,

4. items from the remaining models are likewise put in relation with context factors,

5. the Web engineer extends navigation, view, and presentation model with adaptation specifications and placeholders referring to these specifications, to define adaptation features where desired.

Modeling context in this manner is potentially a lengthy task that can be supported in several ways: reuse, data-mining, and editing tools.

The domain ontology can be read from an existing source, or converted from a prior enterprise information model. Context factors to be used in the model can originate from predefined context catalogs for well-known factors; custom additions of context factors relevant to an application are then made by business experts. Predefined factors can be, for example, categories of time like seasons; for the location category, well-known geographic regions (for example, as defined in ISO3166) or semantic locations (Pradhan, 2000) can be used. The User Agent Profile (Open Mobile Alliance, 2003) is a standardization proposal for defining hardware device capabilities, to be used for content formatting purposes.

Context relations can be automatically suggested by a system performing data-mining of the information stock available in a customer's infrastructure. One approach is to look for co-occurrences and set influence values according to statistical values. For an applied example of generation of a concept network through data mining, see (Staudt et al., 1998). Subsequently to the data mining process, the business expert imports into the model those context relations offered by the mining process that the expert considers to be appropriate, and fine-tunes the model.

If user adjustments to the model are to be possible, a useful display of the context model is needed. Regarding model visualization and editing, the availability of several, potentially connected, ontologies in the context model raises issues of practicability. Potentially, a context relation can consist of as many dimensions as there are categories of context. A corresponding display would presumably be difficult to interpret. However, we expect that individual context relations will typically be connections between two or at most three categories. Furthermore, ongoing research offers new tools that assist in visualizing a multi-dimensional information space; model visualization software is discussed in Section 3.6. In any case, not all ontology elements need be explicitly linked in a context relation in order to be context-relevant (as discussed above), and thus not all elements require a graphic display of potential relations to other dimensions.

Domain knowledge and context factors can potentially overlap. In the scenario presented in Section 3.2.1, it is conceivable that the ontology for the construction market already contains elements that can also be considered context factors: for instance, locations of markets might already be modeled in the domain ontology. In this work's perspective on the engineering process, these should then be considered elements of the context categories as well. To assist the modeling user in such a case, these elements could then be imported into the context categories within the modeling environment. In principle however, context modeling does not require such a strict separation, as the model indiscriminately allows the association of domain or context category elements.

Model Use in the Remaining Engineering Process

Once defined, the context model is available to the remaining steps in the Web Engineering process. One can distinguish between two types of approaches: complete integration or part-wise integration. By complete integration, we understand that all Web application parts are context-sensitive, meaning

all navigation structures and information presented are generated according to context. In a part-wise integration, the application is in some parts context-independent, and in other, selected parts, presents context-sensitive navigation structures and associated information.

It may be assumed that, as a general statement, not all parts of a Web application require a context-sensitive model. In fact, for some aspects adaptation can be counterproductive, resulting in the user becoming disoriented (Nielsen, 1990). Therefore it is an issue of the Web Engineering process to determine in which parts which types of context-sensitive adaptation is useful: the Web engineers specify to use context within the Web software at those places where they deem it useful.

Independently of the degree with which an application is adaptive, context factors must be put in relation with the application domain for these factors to be meaningful in an application: in this light, the decision whether a Web application should be fully or partly context sensitive is secondary with regard to context modeling requirements.

3.5 Application Models and Context

In the following, we detail the models for application development that have been developed in our work. We describe how models for Web applications can be integrated with information about context and how adaptation features can be specified.

All models are described in ontologies, using OWL (see Section 1.1.3). This consistent use of ontologies is advantageous for representation and for implementation. On the one hand, this approach allows for homogeneous usage of context relations on model elements as described in this section; and, on the other hand, a common mechanism for inference and adaptation in a run-time system as described in Chapter 5.

OWL, as is RDF, is an XML vocabulary. Models using OWL can thus be represented in XML; while this information is human-readable, the primary purpose of such information representation is to be machine-processable. For a more user-friendly presentation, the individual models developed in this work are illustrated as RDF-like graphs as shown in (McBride, 2004). In such graphs, an elliptical node represents a resource; rectangular nodes represent literal values. An arc in the graph represents a property of a resource at the blunt end of the arc. The label on the arc identifies the property. The node at the sharp end of the arc represents the value of the property. For simplicity, local names are used in the diagrams (a full name would require a namespace in addition). When a resource or property from an existing schema is used, it is prefixed with the commonly used term:

- `rdf:` identifies names from the RDF vocabulary
- `xsd:` refers to names from the XML Schema data types
- `owl:` refers to a language element of OWL

All types and relations shown without a prefix are defined within the application model as defined in this work. Appendix B gives the precise type definition in OWL syntax.

3.5.1 Domain Ontology

The domain ontology represents domain knowledge; meaning, arbitrary conceptual knowledge relevant to an application, and pieces of content (or, according to type of content, references to pieces of content). The definition of content types is shown in Figure 3.7. Each piece of content can have a `source` attribute containing a URI identifying the content location (for example, the name of an

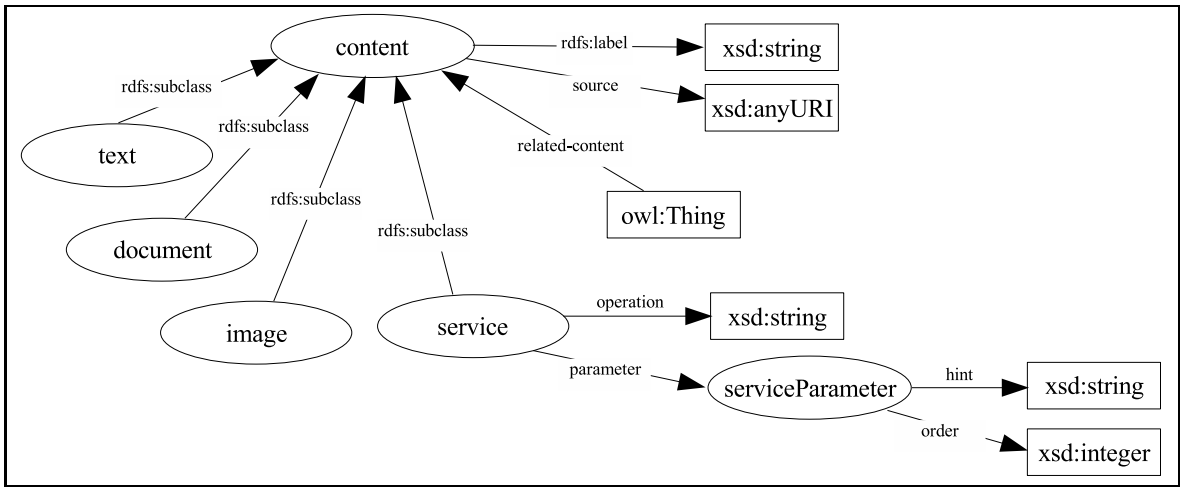

Figure 3.7: Content block types

image file). Content pieces are specified as RDF entries, therefore RDF labels can be associated to them. There can be one such label for each language to be supported. For pieces of text, this label can be used to directly store the content. Any entry of an ontology (that is, any OWL "thing") can be put in relation with a content piece via the `related-content` property. For example, a concept defined in the domain ontology can be put in explicit relation with pieces of content, indicating that these elements are always relevant for that concept.

A service type can specify, in addition, n parameters, each with a sequence number and a `hint`. This hint refers to an entry in one of the model's ontologies, or is an arbitrary name. The intent of the hint is to provide assistance in determining context-relevant information regarding the parameters; however precise interpretation is an issue for the implementation of service incorporation in the run-time system.

Appendix C provides an example domain ontology with content blocks for a prototypical Web application (the prototype itself is described in Section 5.5).

3.5.2 Context Representation

Context factors are organized into ontologies, one for each category of context. The structuring of factors within an ontology can use any available ontology mechanisms and is not subject to restrictions. The context relations are also represented in ontological form, in OWL: this allows for a more straightforward integration with the conceptual model than if the two sorts of knowledge are specified using separate means.

A context relation is represented by a connection between two OWL "things"; this connection carries a value (see Figure 3.8). For model brevity, an unordered n-ary context relation can be represented with the same construct. The definition in OWL syntax of a context relation is given in Appendix B.

An example of a context relation based on the examples outlined in previous sections (see Section 3.2.1 and Figure 3.4) is shown in Figure 3.9. Element `IrrigationSystem` from the domain ontology is associated with element `HeadGardener` from the context ontology User&Role, the association carries a value.

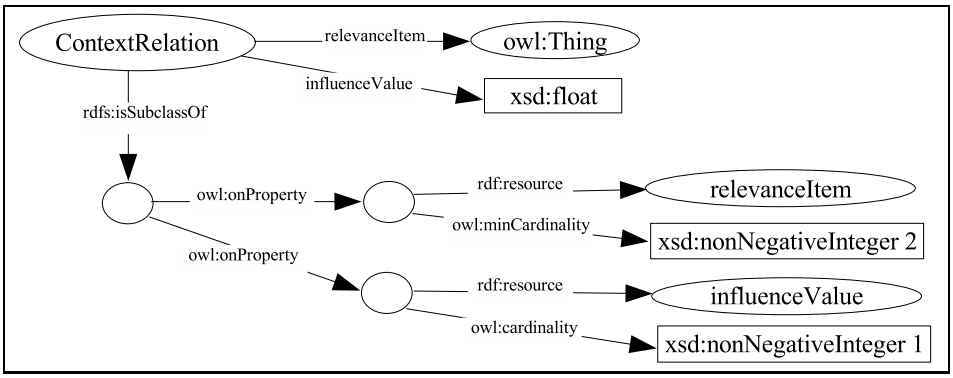

Figure 3.8: Representation of a context relation

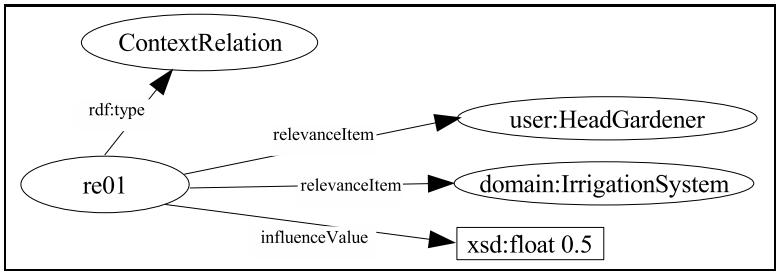

Figure 3.9: Context relation example

3.5.3 Ontology Lookups

The application model allows to define ontology lookups that can then be used for adaptation and for navigation modeling, as shown in the following sections (3.5.4 and 3.5.5).

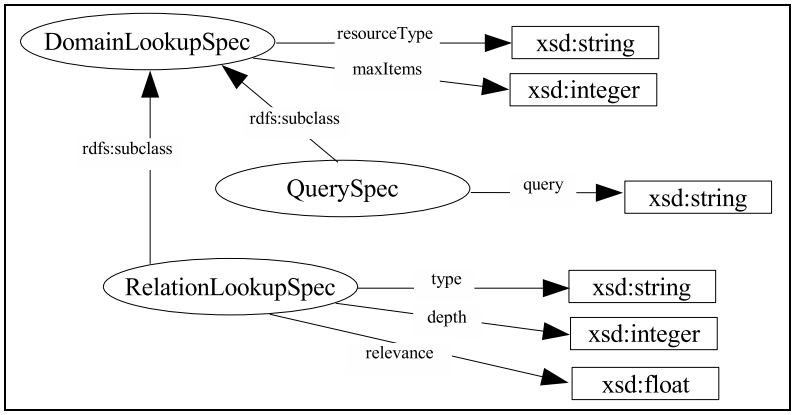

Figure 3.10: Ontology lookup specification

Lookup specifications are used to define a strategy for retrieving elements of an ontology; the properties of such a specification are shown in Figure 3.10. A lookup can be specified either as the following of a specific ontological relation (`RelationLookupSpec`), or as a free-hand query (`QuerySpec`) expressed in an ontology query language such as SPARQL (W3C, 2005).

A `RelationLookupSpec` defines:

- the type of relation in the ontology that should be followed. The type name can be arbitrary, in which case the name can be used for direct relation lookup in the ontology; or a name with well-known meaning can be given, allowing a more intelligent lookup mechanism; for example, “superclass”;
- depth of exploration, meaning to what depth this relation type should be followed;
- the percentage that a found element remains relevant upon each level of relation following. This allows, for example, for ordering of found elements according to relation depth at which they are found.

All lookups can be restricted by defining a maximum number of items and a list of resource types to be retrieved.

3.5.4 Adaptation Specifications

An adaptation specification determines, on the one hand, which sort of information is to be retrieved or generated according to context for a particular usage in the application, and, on the other hand, how the modeled context relations are to be used in combination with ontological knowledge to determine contextual relations in a certain situation. An individual specification can then be used in one or several places in the application models, as shown in the following sections.

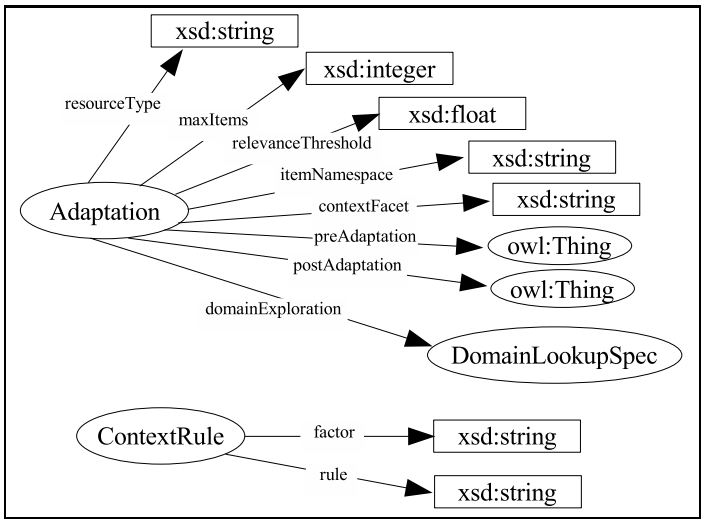

Figure 3.11: Adaptation specification

The properties of an adaptation specification are listed in the following (see also Figure 3.11; for clarity, cardinality properties are not represented in the figure):

- types of items to be computed, or “any” for any type;
- the maximum number of items to be computed;
- context relevance threshold, indicating what minimal relevance an item must have in order to be considered;

- a list of ontologies. If specified, only these ontologies are to be used for looking up context-relevant items. This property can be used to constrain which sorts of application elements are wanted (for example, to differentiate between domain items and presentation items). Ontologies are identified by namespace;
- a list of context categories. If specified, only these categories are to be considered for determining context-relevant information. Categories are identified by namespace;
- a list of `pre-adaptation` items; these items are fixed items from the domain ontology, and are retrieved prior to any context-relevant items, but only if any context-relevant items are found. Analogously, `post-adaptation` items can be specified;
- a list of domain exploration properties, each defining a domain exploration strategy to determine additional context-relevant resources, detailed in the following.

The `domainExploration` property allows to define how items in an ontological relationship with an item found to be context-relevant should also be considered context-relevant, and to what degree (see also Section 3.3.2). An adaptation specification can include an arbitrary number of ontology relationship lookup specifications, as defined in Section 3.5.3.

The definitions for adaptation as shown in Figure 3.11 furthermore allow to specify context rules. A `ContextRule` can refer to an arbitrary number of context factors and associate a formula. The context rule expresses a condition involving these context factors, and can be used in application models which need to express that an action should be taken only when a certain context is satisfied at run-time.

3.5.5 Navigation Model

The purpose of a navigation model for a Web application is to define navigation possibilities between the various elements of a Web application. A navigation model thus has knowledge about relationships between the elements, and can assist in deciding which elements can be visited from the current position of a user in the application.

In a traditional sense, a Web application is a set of pages linked together by hyperlinks, and thus the "navigation model stores the relationship among different web pages" (Zhang et al., 2004). Requirements in this dissertation regarding adaptivity mean that this definition of a navigation model is no longer adequate. Content presented to the user may be selected according to context: a user request may necessitate composition of several resources, where some of these are known in advance and some determined according to context. In addition, which further navigation options are to be presented to a user in a specific situation is not fixed either: some navigation options are pre-determined links (that is, are static), whereas others may need to be dynamically generated, again according to context.

This dissertation therefore defines a specific navigation model, designed to enable a compositional approach to Web application modeling allowing for adaptivity. This navigation model consists of navigation nodes that refer to an entry in the domain ontology via the property `navRelationSrc` (see Figure 3.12). To each node, an arbitrary number of relations can be associated. A relation has a type, to distinguish between further navigation nodes that can be reached from this node, and content blocks that are to be assembled when the user navigates to this node.

The target of a navigation relation can be specified in three different ways. If the target is a single, fixed ontological element, the property `navRelationTarget` is used to refer to this element. If the

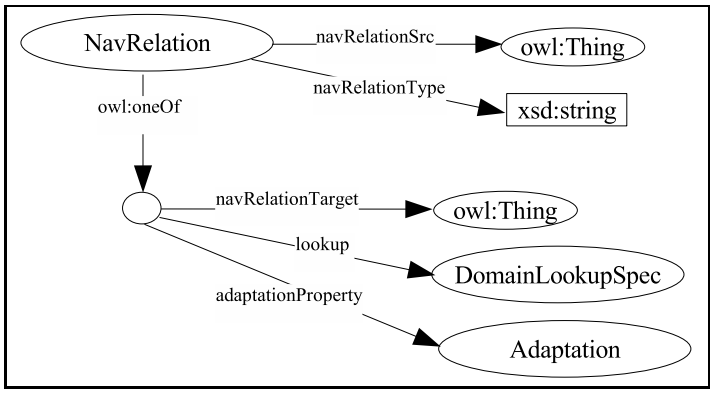

Figure 3.12: Navigation model entry definition

target should consist of a predefined lookup in the ontology (for example, the list of subclasses of the current node), then a `lookup` is used to refer to a `DomainLookupSpec` such as defined in Section 3.5.3. Finally, if the navigation target is not predefined, but should be computed according to context, an `adaptationProperty` is used to refer to an adaptation specification such as defined in Section 3.5.4. This enables adaptation regarding which navigation nodes are available at a certain location in the application, and regarding the content pieces assembled at this node. This adaptation is determined by context, and used at those places deemed appropriate by the application designer.

3.5.6 Modeling for Presentation

The modeling of an application's presentation layer defines which elements are viewable in which situation, and furthermore how elements are presented (that is, which GUI attributes are associated to the elements). The former is defined in the view model; the latter in the presentation model. As these models operate on a similar principle, we summarize them in a common diagram; see Figure 3.13.

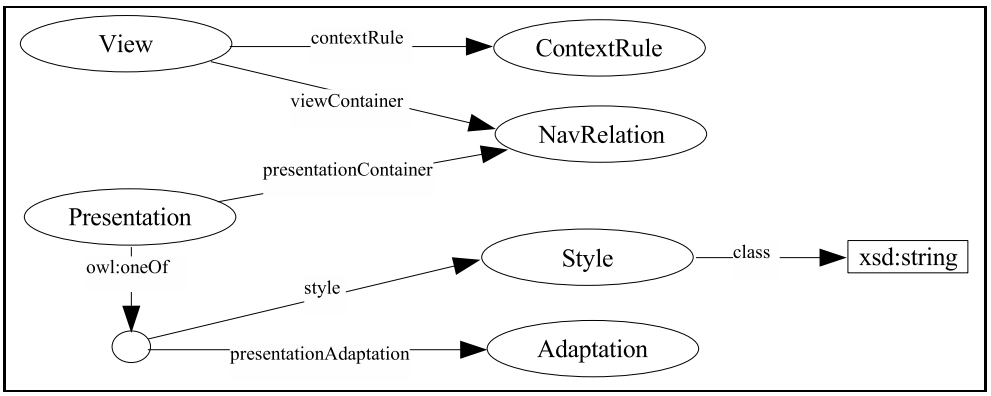

Figure 3.13: View and presentation model definition

In the WISE methodology (see Section 2.3), the view model extends the navigational model to define which elements are to be visible depending on the type of user. We generalize this notion to make a view dependent on context. This means that any condition regarding context state (including user type) can be specified in a view. A view then consists of a context rule (see Section 3.5.4) associated to a navigation relation as defined in the navigation model (see Section 3.5.5).

The presentation model associates style information, that is, information defining GUI characteristics, with elements composing the Web page as defined by the navigation model, and thus each `Presentation` entry refers to a navigation relation. Association of style can be fixed (via the `style` property), or can be determined by adaptation. An entry can have an adaptation property: if so, the style for this entry should be computed according to context. An adaptation specification in this case will most likely constrain the search for context-relevant items to the presentation ontology, in order to compute the presentation model element considered most context-relevant (see Section 3.5.2). If an adaptation was not specified for the presentation entry, or if adaptation at run-time did not yield any relevant style information, then any style information explicitly associated to the entry is to be used.

Instances of `Style` refer to the name of a CSS[3] class; this allows to delegate definition of actual physical GUI properties to a well-known and standardized mechanism.

Use of a concrete presentation model and the effect of presentation adaptation is illustrated, as for the other models, in the prototypical Web application described in Section 5.5.

3.6 Model Visualization and Editing

For user-centered visualization and manipulation of context relationships as outlined in this work, general-purpose ontology editors such as Protégé (Stanford Medical Informatics, n.d.) or SHAKEN (Thomere et al., 2002) are not suited. A particular challenge consists in assisting the user for the task of relating factors from several context categories with the domain ontology. The relation can involve elements from several ontologies and thus be multi-dimensional.

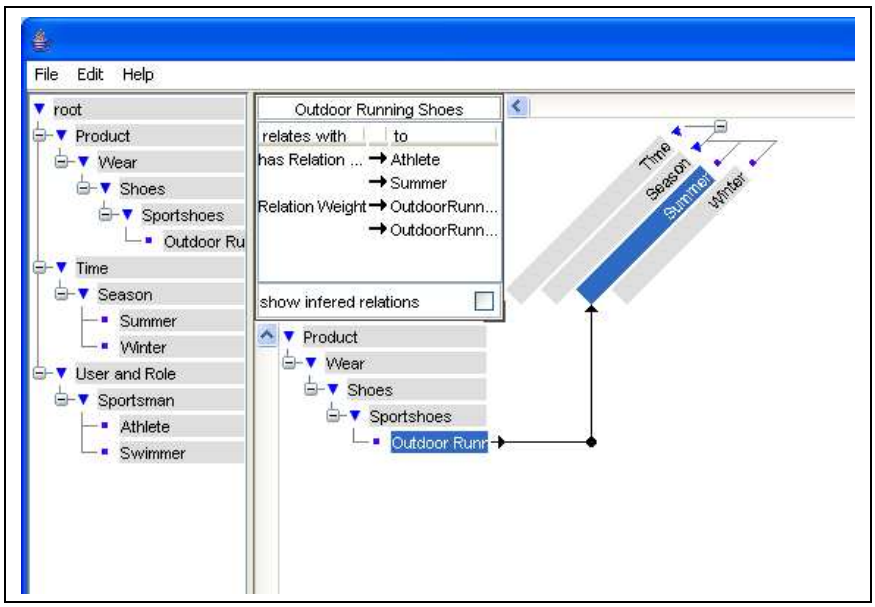

Figure 3.14: Editing context relations using the MatrixBrowser

To support the user in visualizing an existing context model and to allow for editing it, the MatrixBrowser (Ziegler et al., 2002) is one possible solution. The MatrixBrowser is designed to assist in visualizing and editing an information space in two dimensions. Figure 3.14 shows an example of a context model (as understood in this work) being displayed in the MatrixBrowser. The advantage

[3]Cascading Style Sheets, see Appendix A

of the MatrixBrowser's approach is to provide an intuitive visualization of the connection between two hierarchies of information, and is therefore most suitable when the information space is two-dimensional. The MatrixBrowser provides a good overview in situations where a large number of domain elements and relationships exist, but lacks functionality regarding detailed viewing and handling of a single relationship, in particular when the relation involves more than two dimensions.

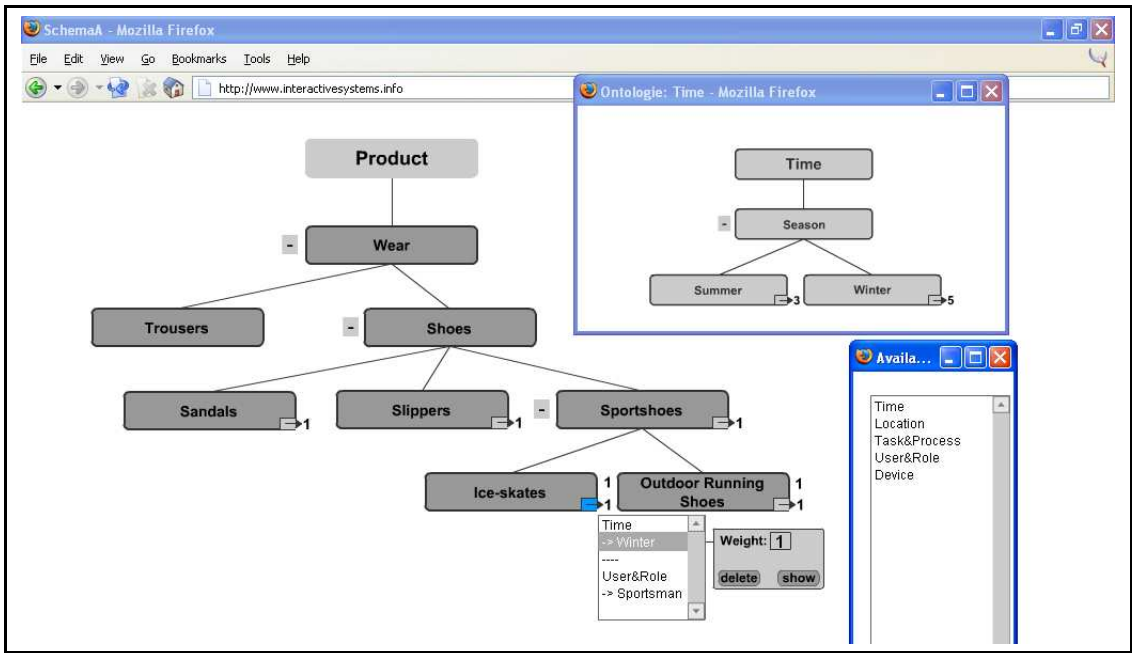

Figure 3.15: ContextBrowser: multi-dimensional context relations

To provide an n-dimensional context editor, a prototypical "ContextBrowser" is developed. Figure 3.15 shows the same context information as that shown in Figure 3.14 displayed in this system. The goal of this prototype is to investigate how n-dimensional context connections can be handled in a user-intuitive fashion. In this visualization, a hierarchy in the domain ontology ("Product") is shown; each element possessing a context relationship displays an arrow and up to two numbers indicating the number of relationships: the first number indicates implicit relationships resulting from a relationship of an element higher up in the hierarchy; the second number indicates the number of explicitly modeled relationships. Clicking on the arrow unrolls connected context categories, showing specifically the related element from this category. The user can switch perspectives to connected ontologies, or have an extract shown in a separate window (in this case, "Time").

3.7 Summary

This chapter has presented a modeling approach to Web Engineering taking context into account. A context model is described, which represents information about the context of usage of an application. Further application models allow to specify a Web application which integrates the context model and contains specifications for adaptation to context within the application.

To support adaptation according to these specifications, a software system is required that makes use of knowledge described in these models. The following chapter discusses architectural issues for such a software system, and proposes an approach for realization.

Chapter 4

A Web Software Architecture for Context

This chapter discusses concepts and techniques for an architecture enabling adaptation to context in Web software.

Section 1.2 introduced the general challenges and aims of the field of research this dissertation is situated in; Chapter 3 described a general purpose context model for Web Engineering and showed how it can be integrated with Web application models. We now discuss the challenges regarding the goal of taking context into account in a run-time system supporting adaptive Web applications. To support this goal, we propose a specific Web software architecture for context. The subsequent chapter (5) in turn details a framework for context implementing the concepts and demonstrating the techniques described in this chapter.

4.1 Requirements for an Adaptive Run-time System

Web applications that go beyond presentation of a collection of static Web pages require dynamic generation. For adaptive applications, each step of such a dynamic generation may require adaptation, meaning that output of each step varies according to context. Dynamic generation of Web applications is supported by several existing software architectures and frameworks. Regarding adaptation to context, some provide a built-in mechanism for dynamically choosing a presentation style depending on the client device; in the terminology used in this dissertation, we say that there is support for adaptation at the presentation level according to the context category "hardware device". In such architectures, adapting generation to further aspects of context is not possible without custom programming. A Web software architecture for context, that would have integrated knowledge about context at all levels of application generation, could allow for using context systematically and for achieving adaptation effects without requiring custom programming (or, at least, limit required programming to the essential).

4.1.1 System Environment

Web software operates according to the *request-response* mechanism, meaning a client (for example, consequently to a user action) sends a request to a server, which must determine the response. More precisely, pages displayed in a Web client (for example, in a Web browser) are requested over the Internet using the request-response mechanism defined by HTTP (see Section 1.1.2). A request is identified by a URL that describes the client's desired resource. This resource need not be an actual file, but can be dynamically generated by the Web application.

In the case of an adaptive application, the response depends on context, and therefore the server must have knowledge of context and process context information. Whereas some processing may be distributed (for example, sensing of the physical environment), the core application generation process must be deterministic as an explicit server reaction to a client request. This means that a *Web server* must possess, for its processing, some form of integrated knowledge regarding available context processing features. Context processing as such can be integrated in the same environment, or be distributed.

In addition to core application generation and context considerations, an application environment may need to integrate packaged application features, which may or may not be independent of context considerations. While the use of such functionality needs to be integrated in the core, the functionality itself is potentially realized in distributed components.

In summary, the architecture needs to fulfill the following requirements:

- the application is generated dynamically;
- the system is context-aware and each generation step must (potentially) be adapted according to context;
- the system contains an interface to context and adaptivity functionality, whereas context processing as such can be integrated, or delegated to distributed components;
- the system allows for integrating arbitrary application functionality; such functionality may exist independently of context considerations.

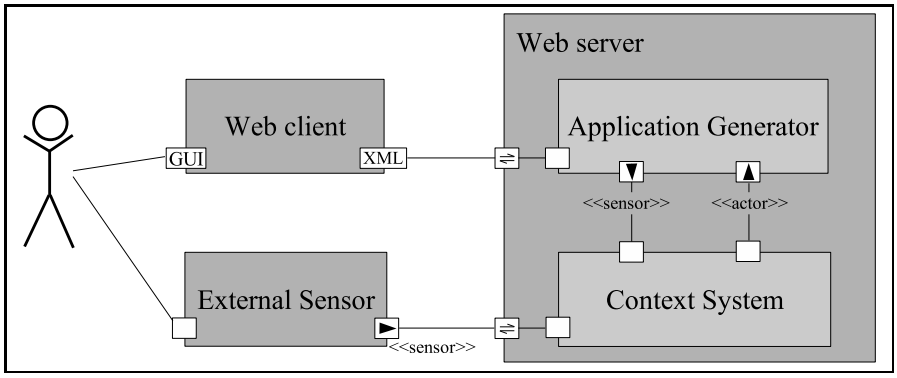

Figure 4.1: Operational environment of the context-aware architecture

The operational environment of a system producing adaptive Web applications is summarized in the UML[1] diagram of Figure 4.1.

An actor (the end-user) operates a *Web client* (such as a desktop browser or WAP-enabled mobile phone) via a graphical user interface (GUI). The *Web client* communicates with the *Web server* to transmit a user request and receive the server's response, in an XML dialect appropriate to this client.

The server response is handled by two subsystems. The *Application Generator* is responsible for core application generation; that is, that part of processing which must exist independently of context considerations.

[1] Unified Modeling Language, see (Object Management Group, n.d.)

The second subsystem in the diagram is the *Context System*. The *Application Generator* functions in a *sensor* role for the *Context System*, as the former provides the latter with user request information. When asked to provide adaptation features, the *Context System* is an *actor* for the *Application Generator*, as the former generates information influencing output of the latter.

In addition, one or more *External Sensor* systems might exist, providing additional context information to the system; for instance, current temperature in the user's environment. Such systems may or may not be actively triggered by a user action, and thus the connection between the user and this system is optional and can not be further specified at this stage. When activated, this system acts in the role *sensor* with respect to the *Web server* system; or, more precisely, to the *Context System* subsystem.

4.1.2 Reference Architecture

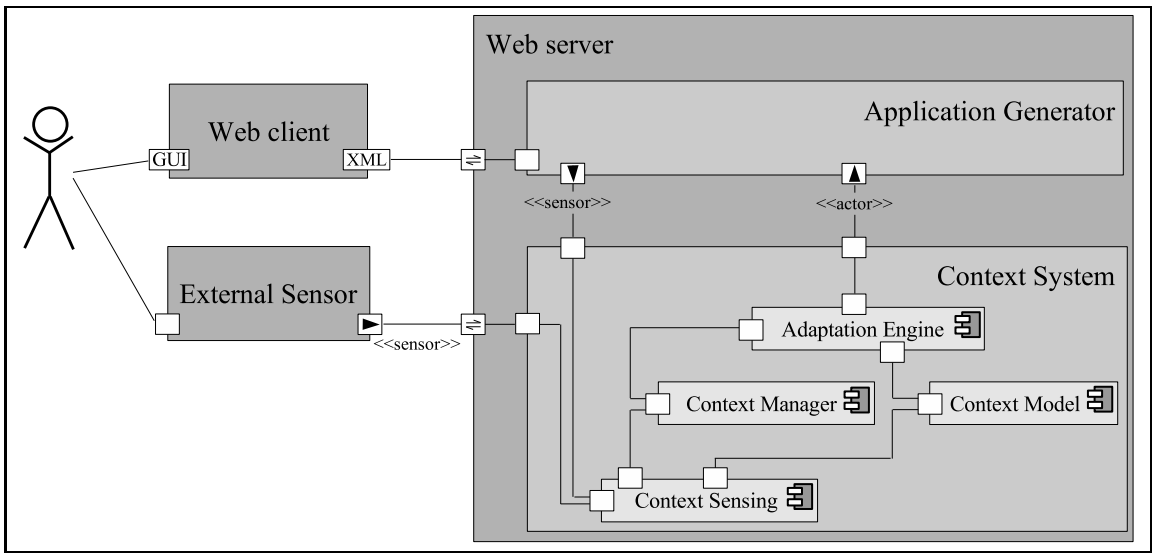

Figure 4.2: Composite structure diagram of the context-aware architecture

The *Context System*, introduced as a subsystem in Figure 4.1, is described in a higher level of detail in Figure 4.2. A run-time context system uses information about context factors and relations established in the modeling process as described in Chapter 3. In addition, the system establishes current context during system operation; we call this information *context state*. The system can then use all context information to provide for *adaptation*.

The *Context Manager* component allows to read and update the context state; the *Context Model* provides access to the modeled information regarding context. The *Application Generator* and any *External Sensor* systems, in their role as *sensor*, provide input to the *Context Sensing* component. This component is responsible for determining context, and may access the *Context Model*. Subsequently to context determination, the *Context Sensing* notifies the *Context Manager* to update context state. The *Adaptation Engine* component realizes the *actor* role of the *Context System*, by providing adaptation features for the *Application Generator*: the former generates output for the latter or guides its output selection. For its operation, the *Adaptation Engine* accesses the *Context Model* to retrieve context elements which may be relevant, and queries the *Context Manager* to retrieve the state of these elements.

The information provided by the Web application's parameters and external sensors is not directly usable as context knowledge. Useful interpretation onto context factors is a task of the context system; specifically, of the *Context Sensing* component. Parameters provided by the Web application can be of two types. The first type is domain ontology elements: the Web application "knows" which ontology elements are currently relevant. The basis for this knowledge is the application's ontology-based navigation, where navigation structure elements are constructed from the ontology, and the user hence navigates through such a structure. The second type of parameters is classical Web application parameters, such as client device parameters, and environment information, such as current time. Such parameters are transformed by the *Context Sensing* component into context factors as defined in the model. For instance, sensing might provide a time context in date, hours and minutes; however in the model only higher-level distinctions of time, such as spring or fall are used, so this parameter is interpreted accordingly. Similar mappings may be needed for the other categories of context as well.

Context knowledge can later be used by the *Adaptation Engine*, that determines which resources are relevant in the current context state. The selected resources are then returned to the Web application, that in turn presents them to the user. Adaptation determination can be guided by various strategies. Notably, given that there is a categorization and structuring of context factors in the model and thus potentially various kinds of relations between context factors and domain elements, a request for context-relevant resources can yield varying results according to the strategy used. The request is more or less granular; typically not only exact matches are wanted, but also resources in relation to these. Relations in the model are weighted, therefore any element of the context space can have a different relevance value in a given context. As a consequence, which adaptation process is to be accomplished by a system and what sort of results are desired must be specifiable. In our work, the system is required to take into account adaptation specifications as defined in Section 3.5.4.

4.1.3 Web Application Elements

As introduced in Section 3.5, a Web application in our approach is built upon resources and their definitions; a run-time system must consequently support the integration and delivery of these resources. These resources can be:

- concepts defined in the domain ontology;
- structured documents or document parts, attached to concepts in the domain ontology;
- arbitrary resources (such as binary documents or pictures), also attached to concepts in the domain ontology, but unstructured (in the view of the Web application);
- service resources. Such services can search for further resources, or represent operational features such as e-commerce transactions. These resources have descriptions defined (or referenced) in the domain ontology.

The specific requirement in this dissertation is that the application offering be adapted through context-awareness: context can influence which resources are offered in which situation, how they are integrated and how they are presented. In a given context (such as described in Section 3.3), the *Adaptation Engine* must thus provide resources deemed to be relevant in this context - we call these context-relevant resources. We distinguish the following types of context-relevant resources (see also Figure 4.3):

- *Navigation* refers to navigation possibilities relevant to this context. For instance, the system can provide links that are potentially of interest.

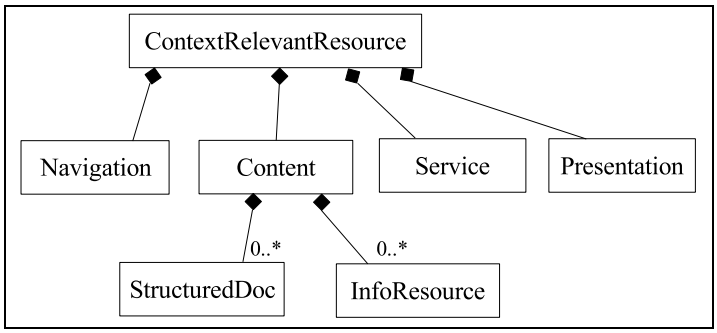

Figure 4.3: Types of context-relevant resources

- *Content* refers to documents or other types of structured information relevant in a given context and that can be retrieved for presentation to the user. The structuring could consist of an XML mark-up such as DocBook (see Appendix A), thus enabling a straightforward adapted presentation through the use of stylesheets. Another type of content element is an arbitrary information resource, such as an office document, that the user can then choose to retrieve and open as a whole.

- *Service* refers to a service relevant to this context. Such a service can be presented to the user, leaving it up to the user whether to execute this service; but such a service could also be transparently executed, for example to dynamically retrieve information without requiring user interaction. Furthermore, a service description for relevant services can be used to dynamically search for available services.

- *Presentation* refers to information assisting the Web application in presenting a resource (or a set of resources) in a manner appropriate to context, for example in a certain graphical style.

In principle, a request for context-relevant resources can result in any number of elements being determined, where each element is of one of the above types. The *Adaptation Engine* must support parameters to constrain this determination; for example, to specify that in a certain situation, only resources of a certain type are to be determined.

4.2 Processing and Architectural Approaches

As introduced in Section 4.1, to dynamically determine a response to a user request, the application generation system must perform several processing steps. The complexity of each step and of the manner in which the steps are orchestrated increases with the goal of adaptation. To manage this complexity, information must be represented in a well-specified manner, and a processing specification is required to define how this information is used to generate a response.

We begin this section by discussing the information representation principle. We proceed by comparing general architectural approaches which can be used to support adaptation.

4.2.1 Information Processing

Two approaches to generation of Web applications can be opposed, content-driven and technology driven. A content-driven approach is based on a description language for describing content. A

technology-based focus can imply technology-biased design decisions, in addition to risks of technology lock-in and deprecation. A focus on content is in principle technology-neutral, yet implies on the other hand that the way content is to be processed must be clearly specified; and that a software architecture be established that supports this focus.

Content-driven	*Technology-driven*
XML and XML-based vocabularies	PHP
OWL	Active Server Pages (ASP)
XSL	Java Server Pages (JSP) and JSP templating such as Struts

Table 4.1: Content-driven vs. technology-driven representation

Table 4.1 gives an overview of possible elements in these approaches. XML is a technology-neutral basis for representing information, as introduced in Section 1.1.3. OWL (an XML dialect) allows to define ontological knowledge, such as relations between terms used in an application. Development of corresponding Web applications based on this information, in particular adaptive applications, can be supported by XSL, a further set of XML-based languages (see Section 2.2.4). XSL allows to specify processing steps of XML data and is thus one possible means to manipulate XML-based representations of knowledge.

This content-centered view, while placing the primary focus on content descriptions and transformations of this content instead of focusing primarily on a technology, does nonetheless require solutions to the following issues:

- specification of XML content;
- specification of dynamic content;
- definition of the logic determining what content is to be retrieved, and how content is to be retrieved and assembled;
- handling of logic for user flow and navigation in the Web application;
- presentation of content;
- management, configuration and editing of the Web application and its contents.

Solutions to these issues require architectural design decisions and, in part, do become related to technology. Chapter 3 explained how domain and context information can be specified, and furthermore how Web application models can be used to define application composition and behavior. The following in turn discusses concepts and techniques for integrating and making use of such models in a run-time system.

4.2.2 Architectures for Adaptive Systems

As introduced above, to achieve adaptive systems, there is a general need for an information processing architecture in which context can influence processing, and various system parts exist to accomplish this processing. The question then is what core architecture(s) to use to support such systems, how to coordinate system parts and how information is exchanged between those parts.

Table 4.2 provides an overview of possible general architectural approaches addressing these issues.

	processing	*information exchange*
XML-publishing framework	integrated	integrated (SaX-based)
Web Services	distributed	network (XML text)
CORBA	distributed	network (serialized objects)
JavaSpaces	distributed	shared
adaptive agents	distributed	network

Table 4.2: General architectural approaches for adaptive systems

An XML-publishing framework is a framework that includes software components and processing configuration features to specify, and implement, XML-based publishing. This means that, in a system based on such a framework, information is represented in XML, and information exchange between components realizing the core processing is XML-based. As the system is integrated, information transfer between processing steps can occur using an integrated technique, such as SaX events[2]. With this technique, intermediate XML representations within a processing sequence need not be fully built. Processing is instead based on the occurrence of specific XML elements or attributes; such an occurrence is called a SaX event. An example of an XML-publishing framework, Apache Cocoon, is introduced in Section 2.2.4.

Web Services and CORBA, introduced in Section 2.2.3, are examples of distributed processing architectures. Information exchange occurs over the network, therefore all information necessary for processing must be exchanged either by explicit parameters to the components, or by the components themselves calling a further component to access information such as the system's context state.

The particularity of an approach such as JavaSpaces (Flenner, 2001) is to provide a central readable and writable space in which information is represented, while still supporting distributed components. The information in this case is represented as Java objects, so any information represented in XML first needs to be fully parsed into corresponding Java objects.

Adaptive agents are used for computation in situations where uncertainty is present; typically an agent forms an entity that can simulate a natural phenomenon (Resconi and Jain, 2004). Adaptation in this area means that the agents themselves adapt (evolve) over time.

Design decisions on which type(s) of architecture to use to build a system and its subsystems depend on the exact purpose of the systems, and thus architectures can not be assessed independently of such a purpose. Distributed systems are potentially more powerful, as they can harness the power of multiple machines to solve a given problem; whereas building a system on distributed subsystems can result in slower system response: "all distributed systems incur a significant overhead when compared to their serial equivalent: communication" (Sheil, 2003).

4.2.3 Component-based Software Engineering and Adaptation

Basing development of applications on a component approach has the following goals: a unified management of the parts of an application, the possibility to exchange a component with another, and the possibility to define what an application is through a strategy of selection and aggregation of components.

The precise understanding of what a component is varies significantly in literature. In an architecture for Web applications, general-purpose components may exist to implement the architecture; and furthermore individual applications may use components to realize domain-specific functionality.

[2]Simple API for XML, see (Skonnard and Gudgin, 2001)

Using components in this manner can be considered a form of component-based software engineering (CBSE, see Section 2.2).

The notion of adaptation can be used in various perspectives for CBSE. Adaptation in our sense is defined by the variability a run-time system can provide because it is context-aware, in order to modify system behavior according to a situation of usage. Adaptation in this sense can come into play within component-based systems at various levels:

- component selection: context can influence which components are presented to the user;
- component parameterization at run-time: once the decision has been made to use a component (either by the system or by explicit user selection), context information can be used for automated parameterization;
- the rendering of a user interface for the component can be adapted according to context.

Related to the notion of component offering is component deployment, specifically "context-aware deployment" (Ayed et al., 2004): there, selection and configuration of components is handled dynamically by a context pre-processor that generates the deployment plan according to context.

Adaptation in the sense of this dissertation has a different meaning than sometimes used within CBSE as "component adaptation" (Heineman and Ohlenbusch, 1999). In that sense, component adaptation is the manual process when a software engineer uses an existing component and adapts it to new requirements.

Fundamental questions to be addressed in a CBSE approach, according to (Schneider and Han, 2004), are (i) what is a software component; and (ii) how are the software components composed. Section 4.3 offers definitions of components as used in the Web software architecture for context.

4.2.4 Processing in the Cocoon Framework

As discussed in Section 4.2.1, a general-purpose architecture for Web software must address fundamental questions such as how content is integrated and processed, and how additional program logic is specified and in turn integrated. Architectural design patterns can be used (and developed), while best-practice knowledge is gathered over time. Ideally this knowledge is taken into account by a general-purpose, widely-used framework, that enables to develop applications of higher "quality". What precisely is to be understood as quality depends on the framework and the application domain it is typically used for.

In this section, we discuss processing principles in Apache Cocoon (see Section 2.2.4), and introduce how Cocoon can be used as a basis for adaptive Web applications. For detailed explanations regarding Cocoon techniques, see (Moczar and Aston, 2002).

As introduced in Section 4.1.1, Web clients make requests to the Web server, in form of URLs. Cocoon, as a basis for implementing server functionality, must process such requests and generate a response. Which request is handled in which manner, resulting in which sequence of processing steps, is defined in a Cocoon *sitemap*. A Cocoon sitemap consists of *pipelines*, where a pipeline is the sequence of processing steps to take to respond to a request. Typical steps are:

- matching the request: the pipeline decides whether it should handle the request. This decision can be based on various factors such as URL pattern, presence of certain parameters, or client device type.

- generating XML data: one or more XML data sources are read, either from files or from remote XML-based services. XML structures can also be dynamically generated by a custom software component.
- transforming XML data: there may be several transformation steps, between various XML formats. The final transformation step is typically the transformation of XML data into a format which a user device, for example a Web browser, can use to provide a user-readable view. This format can be an XML dialect such as XHTML, or can be an arbitrary, non-XML format.
- serialize XML data into a HTTP response that is sent back to the Web client.

Each step can be handled by a built-in Cocoon component or by a custom component extending the Cocoon framework if additional features are required. Custom logic can further be added between any two steps in form of an *action*, which is Java program code that can be used to further influence pipeline handling, and to trigger back-end processing, such as to prepare data that pipeline steps can then access. Components and their integration in a pipeline can thus be thought of as "active Lego bricks for XML manipulation" (Mazzocchi, 2002).

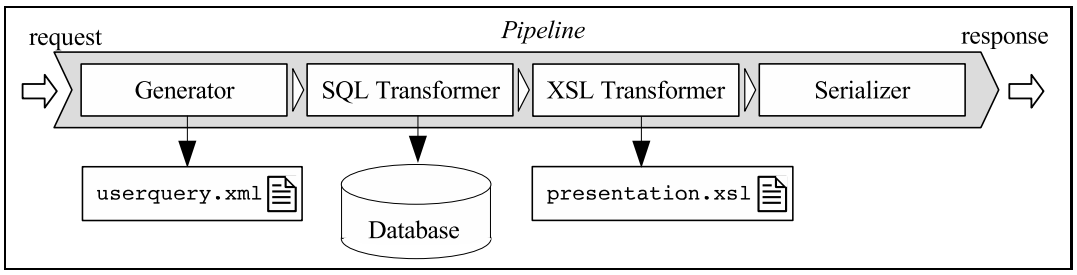

Figure 4.4: Cocoon pipeline example

For illustration, consider the pipeline example presented in Figure 4.4, which is inspired by (Punte, 2002). In this pipeline, content is generated dynamically (by querying a database) and rendered dynamically (by performing an XSL transformation). The pipeline first generates XML data; in this example the data consists of an XML structure defined in a file `userquery.xml` containing, for the dynamic parts, SQL statements indicating how to read this information from a database. The *SQL Transformer* is then responsible for reading these statements, calling the database to retrieve the data, and consequently replacing the SQL statements with the actual data. The next step in the pipeline is to transform the XML data into XHTML; this transformation is achieved by applying an XSL stylesheet (`presentation.xsl`). Finally, the pipeline serializes the data into an HTTP response.

Listing 4.1 shows how such a pipeline might be specified in a sitemap.

```
<map:match="*.html">
   <map:generate src="userquery.xml"/>
   <map:transformer type="sql">
      <map:parameter name="use-connection" value="mydatabase"/>
   </map:transformer>
   <map:transformer type="xsl" src="presentation.xsl"/>
   <map:serializer type="html"/>
</map:match>
```

Listing 4.1: Cocoon pipeline configuration example

4.2.5 Using Cocoon for Adaptivity

Cocoon, by conception, supports design patterns and contains mechanisms appropriate to serve as foundation for adaptive Web application generation.

Cocoon follows an explicit level of separation of concerns typically understood as the Model-View-Controller (MVC) design pattern, meaning the separation between content, logic and style. In Cocoon, the XML data is the model, views are achieved through XSLT, and the sitemap (including components used by the sitemap) is the controller. This separation allows to address adaptation at the various levels of a Web application (see Section 3.1) in a manner consistent with the architectural principles of Cocoon. The MVC pattern is increasingly accepted and thus supported by various Web application frameworks. The particular added-value of Cocoon for the goal of an adaptive architecture is to be built upon a well-defined component-based processing model. Each pipeline processing step is handled by a component with a clearly defined processing role in the pipeline (such as, an XML transformation step). Each step is thus realized by a specific component, that is either built-in to Cocoon or a custom component, realizing, on the one hand, a generic processing concern of a pipeline and, on the other hand, a functional concern of the application (for example, binding of a database in the pipeline).

The generation of an application as a composition of resources is supported by Cocoon sitemap mechanisms, notably aggregation and recursivity. Aggregation means that individual parts of the generation process can be described in their own pipeline specifications, and results aggregated in an enclosing pipeline. Pipelines can be recursive; since a pipeline step can call any other pipeline step via URI (and provide parameters for this call), the pipeline can also call itself: for example, if an adaptation process has resulted in too few elements being retrieved, the same pipeline segment can be called again with different parameters.

Regarding adaptation to user device characteristics in particular, Cocoon's browser selector, a sitemap component, can be used to match user device information in order to select an appropriate XSL stylesheet. For example, a pipeline can be set up to deliver an XHTML version of a document if the user is using a desktop browser, whereas a user viewing the document with a WAP phone would receive a WML version of the document. The way in which the document data is generated is identical in both cases, and Cocoon then selects the appropriate XSL stylesheet for the user device to transform the document into a viewable representation. In other words, Cocoon provides built-in mechanisms to support some level of adaptation to device characteristics; contrasting with Web applications based, for example, on ASP or JSP scripts, where the developer needs to explicitly program such adaptation within the script itself. Using user language preference setting within delivery mechanisms and handling user session parameters are further "out-of-the-box" adaptation tools; such features are common to this type of framework - see, for example, Struts as used in (Bellas et al., 2004).

The above-mentioned features are helpful to enable adaptation, but as such not sufficient to implement the goals outlined in this dissertation. However, Cocoon, as a software framework, is a suitable base for further adaptivity services: on the one hand, all information and logic is XML-based, and this XML can be dynamically generated. This concerns the XML structures describing the content being processed, and furthermore the transformation stylesheets (XSL) and logic sheets (XSP) that can also be dynamically generated and dynamically chosen for execution and parameterized. On the other hand, the Cocoon components themselves are extensible and replaceable: an adaptation service not readily provided by XSL and XSP mechanisms can be implemented as a Cocoon component and thus made part of the framework in a manner transparent to an individual application using the framework. The following section discusses how components can be developed to augment Cocoon functionality, and how they can be integrated into the Cocoon environment.

4.2.6 Components in Cocoon

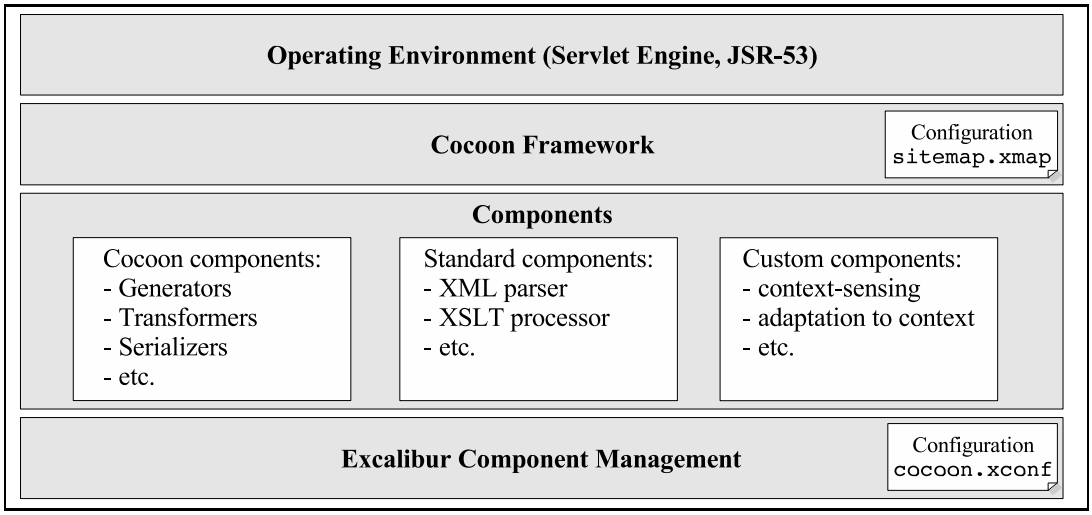

Figure 4.5: Cocoon component architecture

The Cocoon framework uses, as underlying framework for its implementation, Apache Excalibur (see Section 2.2.4). Excalibur is a general-purpose component framework; Cocoon hence follows a component-based architecture (see Figure 4.5), allowing additional components to be integrated. The Cocoon environment defines roles for components usable in a pipeline, such as "Generator" and "Transformer". Cocoon contains a set of built-in components implementing these roles; for example, the *FileGenerator*, which reads an existing file in order to generate an XML stream for further processing in a pipeline. Furthermore, standard components, such as an XSLT processor, can be used. In addition, the Cocoon environment can be augmented by custom components. Such a component is either a custom implementation of a pipeline role, or a component realizing software functionality which can then be called from within a pipeline step.

We now describe how a component is specified and assembled in Cocoon and Excalibur. In Excalibur's terminology, a component is called a *service*, and plays a specific *role*. This term refers to an interface describing the functionality provided by this service and one or more implementations for this functionality. The term "service" in Excalibur's environment is not to be confused with a "Web Service". A component in Excalibur's sense is an interface definition (in Java), one or more Java classes implementing this interface, and corresponding configuration. A Cocoon component is an Excalibur component that additionally conforms to one or more Cocoon contracts (specified as Java interfaces) and that can be configured via Cocoon mechanisms.

Component Setup and Lookup

The control of Excalibur components happens according to the inversion of control (IoC) design pattern. This means that configuration and setup information is provided to the component by the container, instead of the component asking for this information. Furthermore, this means that the component container is responsible for managing the component's life-cycle. For this to work, the component implements the life-cycle interfaces it shall conform to, for example `Configurable` or `Poolable`. The container is responsible for determining which interfaces the component implements, and for calling the corresponding methods at the appropriate time of the component's life-cycle. For

example, when the software requests to use a component, the container is responsible for creating an instance if necessary and for initializing it before returning the instance to the caller.

When an implementation for a specific role is required, it is looked up via the *service manager* (provided by the component framework), by providing the role name. If a role can have several implementations, these are accessed via a *service selector*. The role configuration (given in a corresponding file) in this case must indicate that this role is implemented by a list of components, and not by a single component. Individual components are configured in the component configuration file, by listing their names and the classes implementing them; this list is configured within an entry for the role name. In this case, the caller looks up a *service selector* for that role, and then proceeds to request a particular component.

Implementing New Components

A new component must define, as a Java interface, the role that this component implements. The role name must then be specified in the role configuration file, and any additional configuration required for the component(s) implementing the role is specified in the component configuration.

Components can be defined to fulfill general-purpose functionality, meaning functionality independent of a particular Cocoon processing step. Such a component is then looked up by any other component in the application scope, via Excalibur's `ServiceManager`.

Components can also be defined to be directly usable in Cocoon pipelines. For this, a component must implement specific Cocoon interfaces (in addition to the relevant Excalibur contracts), according to the desired role in pipeline processing. Such roles are notably:

- a `Transformer` is an XML-processing step within a pipeline. The `Transformer` is, at the same time, `XMLConsumer` and `XMLProducer`, meaning it reads an XML source (via SaX events) and transforms this input into another XML format. When a pre-determined structure is to be transformed to another well-determined structure, the transformation can be specified using XSLT stylesheets, which can be processed by the built-in XSLT transformer. When a more specific or less predictable transformation is to occur, a custom transformer can be implemented. Within a custom transformer, any sort of Java processing can be defined before XML output is generated. In the pipeline example from Section 4.2.4, the *SQL Transformer* waits for a specific XML element that contains an SQL statement. The statement is then executed (via JDBC[3]) and the result transformed to XML. The next pipeline processing step needs to know how this transformer structures this XML, in other words, in what form the database result is represented in XML.

- a `Generator` reads a source and produces XML based on the data in this source, thus allowing for later pipeline processing of this data. A custom generator typically accesses a custom source of information that is not represented as XML and produces an XML representation of it.

- an `Action` encapsulates arbitrary logic processing that is to be triggered by a pipeline event, but which is not intrinsically related to XML processing.

- a `SourceFactory` can implement a custom protocol. This is of use when the same data needs to be accessed in various components, and where this access needs to be kept flexible, for example to allow for remote access or to support a proprietary mechanism necessary for access.

[3] Java Database Connectivity, see Appendix A

> Implementing a custom protocol within Cocoon eliminates duplication of code in information access and allows for specifying sources with that protocol directly within the Cocoon pipeline.

To be used by regular Cocoon pipeline processing, custom components must further be configured within Cocoon configuration files. For details regarding Cocoon processing of components, as well as Excalibur configuration and life-cycles, refer to (Ziegeler and Langham, 2002). For an experience report on how custom components can be developed and how these components and proprietary XML-based protocols can be integrated with Cocoon, see (Culshaw, 2003), who developed a travel-agency Web application integrating several (existing) enterprise applications.

4.3 Principles of the Web Software Architecture for Context

We now discuss how content-driven processing and component approaches can be combined to form a Web software architecture for context.

The system's general requirement is that it can dynamically generate an application based on a set of models. Specific requirements are that the system must allow for awareness of context and allow for adaptation at various application levels: navigation, content and presentation. The system must thus take into account various concerns. Ideally, the way each concern is taken into account is extensible, to simplify customization and reuse. Handling the challenges each as its own concern follows the design principle of *separation of concerns* (SoC, as introduced in Section 3.1).

The first stage of separation of concerns was motivated in Section 4.1.1 and shown in Figure 4.1: a *Context System* handles context sensing and processing, and is separated from an *Application Generator*, that functions, at its core, independently of context considerations (though its output can be influenced by calls to the *Context System*).

The second stage of separation of concerns requires a different focus for each of these two subsystems.

Regarding core application generation, requirements are to read application models and generate an output appropriate to the current request. In the approach to Web development followed in this dissertation, application and context models are represented in XML (see Section 3.5), respectively in OWL, an XML-dialect. In addition, the output of the dynamically generated application can also be represented in an XML-dialect, such as XHTML for Web browsers or WML for WAP-enabled mobile phones. Further final output formats such as HTML for backwards compatibility in Web-browser technology or PDF for document viewing can be readily produced by stylesheet transformation from an XML structure. The generation process as such is divided in the above-mentioned levels, navigation, content and presentation. Application generation at each level is provided by individual system parts (i.e., components), thus achieving a second stage of SoC. These generation steps produce intermediate XML representations that need to be sequenced by the surrounding system. In conclusion, as the core generation must provide a well-determined and timely response to the user, an integrated system is required to implement the *Application Generator* subsystem. In an XML-based environment such as outlined in our work, an XML-publishing system provides built-in features to assist in this task, and is thus used to implement this subsystem.

The *Application Generator* is thus located within an XML-publishing system, which must then be interfaced with the *Context System* to realize the *sensor* and *actor* roles described in Figure 4.1. Consequently, components must be designed within the XML-publishing system to realize these interfaces; whether the processing required by these components is fully integrated, or whether the components act as proxies, delegating actual processing to distributed components, is a separate design issue. Which design is appropriate depends on the exact requirements of a specific environment.

The primary concerns of the *Context System* are to ensure context-awareness and to provide adaptation features. Context-awareness depends on parameter sensing and, potentially, on additional sources of information given to the system. A general-purpose architecture for context can support context-awareness, but specific context sensing varies according to the application. If external sensors exist (see Figure 4.2), the *Context Sensing* component must provide an external interface from the Web server environment. This interface can then either be URL-based and use the same entry point as user requests, or exist as an independent network service (implemented, for example, as a CORBA service). Adaptation features in turn are offered by the *Adaptation Engine*, that may be used by each component within the *Application Generator* requiring such features. Implementation of the *Adaptation Engine* may rely on external, distributed systems, or may be fully integrated. One possibility for integrated operation is to implement adaptation rules defined in an application model such as shown in Section 3.5.4. If distributed components are used for implementation of the *Context System*, and need to have common access to the *Context Manager*, the latter can be implemented with a system such as JavaSpaces.

4.3.1 Processing Model

This work views a Web application as a collection of resources. This covers any "thing" that is relevant to the system, be it a resource such as a document or a document part, or be it a resource representing an operational feature; the types of resources are discussed in Section 4.1.3. All resources are related to concepts in the domain ontology and potentially also to one or more context factors. A service resource can return n resources for display. An adaptive "resource" in turn is a placeholder, and can also represent n resources once processed by the *Adaptation Engine*. Resources returned by an adaptation process can be of any type.

This notion of what an application is requires a processing model to determine how the application is generated, and furthermore an architecture supporting this processing model.

Context-Leveraging Stack

This dissertation proposes a system architecture supporting dynamic generation of Web applications, where XML is the foundation and XML-based processing is used for page generation and context adaptation. Information about the domain and context is modeled and represented as in Section 3.5.2; further models determine how an application is composed, where adaptation takes place and which strategies to use. These determinations are thus guided by descriptive modeling and do not require explicit software programming.

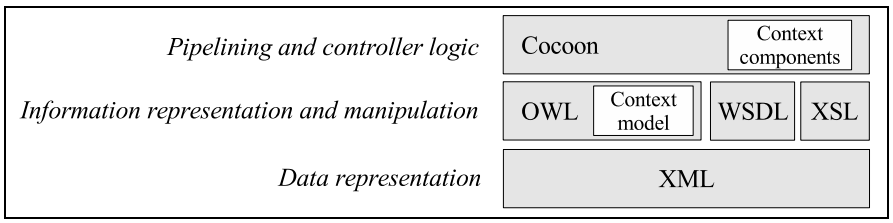

Figure 4.6: Context-leveraging stack

Conceptually, the combination of XML, XML-based standards and an XML publishing system such as Cocoon augmented by components for context can be termed a context-leveraging stack (see

Figure 4.6). In the context-leveraging stack, XML is the foundation allowing for data representation. Domain and context models are represented in ontologies, using OWL. Web Services are used for dynamic, domain-specific functionality; their interface is represented in WSDL. Web Services known to the system are furthermore labeled in the ontology, to provide a hook with domain and context elements. XSL, respectively its subparts XPath and XSLT, is used for immediate information manipulation: XPath for accessing information, and XSLT for transforming it. Sequencing and parameterization of XSLT stylesheets is handled by an XML publishing system (Cocoon, see Section 4.2.4).

By accessing the appropriate entries in the models (that are all specified in OWL), XSL can be used for the core of the dynamic generation of the Web application, without requiring additional programming. By core, we understand the selection and composition of Web resources, not including features of context adaptation and service incorporation. To support these features, placeholders are generated in XML syntax that can then be processed by custom components extending the Cocoon framework. Taking context into account is thus handled as a separate concern; generation of the Web application works independently of context, yet can potentially be affected by context in each generation step.

Processing by XML Transformations

We term *pipelined transformational approach to adaptation* the approach in this work for achieving adaptation by using the flexibility provided by sequencing XML transformations through controller logic. Conceptually, the approach is similar to (Fiala et al., 2004):

> According to the user/platform profile ...[document generation] is subdued to a series of XSLT transformations, each considering a certain adaptation aspect (...). Instead of evaluating all adaptation rules "at once", the generation process can be divided into more steps in order to reuse partially adapted documents for similar requests.

In this dissertation's approach, each step of the generation process is an XML transformation, whereas this transformation can be specified by XSLT instructions or by a custom component performing the transformation (called, in the following, a "transformer component"). If the functionality a transformation step must provide is well-determined, the transformation step can be specified as an XSLT stylesheet, as XSLT is a general purpose language for transforming some (possibly dynamic) XML source into another XML structure. If instead functionality required in a transformation step is more complex (such as, to integrate arbitrary adaptation results), a custom transformer component is used.

This approach is a foundation for composition and generation in the sense of component-based software engineering (CBSE). The CBSE rule-base here is then a set of transformations augmented with surrounding logic (the pipelines) to sequence several transformation steps, affect their selection and parameterization according to pre-determined rules as well as contextual knowledge. A transformation converts one form of XML into another, where output can be augmented with information retrieved from additional sources (such as resources resulting from adaptation). The rule set (again, in the CBSE sense) thus consists of components for context, XSLT statements (grouped in XSLT stylesheets) and pipeline definitions:

1. pipeline statements make resource selection and appropriate XSLT stylesheet selections, and parameterize resources and XSLT stylesheets;

2. XSLT stylesheets provide rules for template matching and conditional processing. For example, the `xsl:choose` statement can be used to determine which parts of an XML resource are to be

selected, depending upon context;

3. after resource selection, those resources which are dynamic (the services) may require execution: this can be achieved either by a custom transformer component, or by further XSLT stylesheets containing rules specifying how to transform the dynamic component description into the concrete service call. The call itself is triggered by the pipeline entry, and the result made available to the next steps in the pipeline;

4. at the end of the pipeline, contents to be rendered in the GUI are aggregated. The GUI is rendered by one or several XSLT stylesheets. The selection of stylesheets can again depend on context. XSLT statements transform XML content into a format appropriate to the end-user device.

The pipeline approach can be supported by a framework such as Cocoon. As discussed in Section 4.2.5, Cocoon contains built-in mechanisms for handling of XML- and XSL-based pipelining, and furthermore can be extended with components to augment pipeline processing capabilities, as discussed in the following section.

4.3.2 Architectural Components

The Web software architecture for context proposed in this dissertation extends the Cocoon framework with a set of components, on the one hand to support context-awareness and adaptation functionality, and, on the other hand, to allow a generation process in a pipeline which allows to integrate adaptation features for each phase of application generation, and furthermore to incorporate operational features specified in the application model.

We call these components *architectural components*, and distinguish between components for application generation and components for context. Some of this functionality is generic, meaning independent of a specific application; in addition a particular application or application domain may require custom context components to extend generic context functionality.

In the sense of CBSE, service resources (see Section 4.1.3), providing an operational feature for an application, should also be called components; we term these *domain-specific components*.

Component type	*Main purpose*	*Basis for design*
Component for context	context interpretation and adaptation functionality	Cocoon (Excalibur)
Component for generation	adaptive application generation and incorporation of operational features	Cocoon pipeline roles
Domain-specific component	application- or domain-specific functionality	Web Services

Table 4.3: Component type overview

Table 4.3 summarizes the types of components. Domain-specific components are discussed in detail in Section 4.3.3; here, we provide a conceptual definition of the architectural components. Concrete components are proposed in Chapter 5.

We define components for context as follows:

1. a component for context fulfills a specific concern of context-awareness or of adaptation to context;

2. composition and usage of such a component is guided by the principles of Cocoon's underlying component framework, which is detailed in Section 4.2.6.

Basing the implementation of such components on an existing component framework (Excalibur) provides a conceptual setting for the components and furthermore facilitates development and run-time usage. In addition, as Excalibur is the underlying component framework of the Cocoon architecture, these components can be used transparently in this architecture to augment it for desired functionality, namely context-awareness and adaptation.

We define components for generation as follows:

1. a component for generation implements a step in the application generation process, that is, in the generation of a response to a user request, and is in addition defined by the pipeline processing role it plays according to the Cocoon processing model;

2. composition and usage of such a component is guided by the principles of Cocoon sitemap and pipeline configurations, see Section 4.2.6.

4.3.3 Domain-specific Components

We define domain-specific components as follows:

1. a domain-specific component is a service resource, providing an operational, potentially interactive, feature. The component is described by a service description, that is defined (or referenced) in the domain ontology;

2. services are composed as any other Web application resource, by reference in the Web application model. This reference is either explicit, or results from the service being considered context-relevant in the scope of an adaptation specification.

The usage of services as (domain-specific) components follows the service-oriented philosophy as outlined, for example, by (Cervantes and Hall, 2004): as in any type of component-orientation, "applications are assembled from reusable building blocks"; in the service-oriented approach "the building blocks are *services*". Cervantes and Hall describe in addition how a system can adapt to availability of a service at run-time, and autonomously look for alternatives. In this dissertation, the focus is instead on context-sensitive selection and incorporation.

As the related field of hypertext application development sometimes uses the term "component" to refer to a navigation node (Halasz and Schwartz, 1994), some approaches in Web Engineering also use the term in this manner. For example, in (Wissen, 2004a) a component is any type of resource available to a Web application (see Section 4.1.3): a "concept" in the domain ontology can be used as navigation node, and as such is considered an application component. This is however not the granularity expected from a component in the CBSE sense (see Section 2.2), and therefore care must be taken when using the term component. While our work does not use this term for application resources other than the services, there exists nonetheless a common perspective on all resources that allows for a generic, homogeneous way of associating context with the resources used in an application, independently of the type of resource. In this perspective, a Web resource represents one part of the information or service offering provided to the user. Context can then influence which resources are selected and presented to the user; and ideally context can also provide a configuration for those resources that are (domain-specific) components.

Web Services as Domain-specific Components

Web Services (as defined by the W3C, see Section 2.2.3) can be used to define and implement the service offering. The Web Services standard defines a component approach which has similarities to previously existing approaches; see (Schneider and Han, 2004). This particular approach is well-suited for the goal of context-aware application generation based on the composition of resources. This has conceptual as well as technical reasons. Conceptually speaking, we consider a Web application to be an integrated point of access for a user to a diverse set of resources (see Section 1.1.2). These resources are either fragments of information, or services. The resources need not be intrinsic to the Web application, but may be available somewhere, for example in a company IT system, or in the Internet as a third-party offering. The notion of Web portals as it is sometimes used - see, for instance, (Wege, 2002) - shares to some extent this vision of what a Web application is. However, the term "portal" can evoke a specific portal server technology as described in Section 2.2.1, which is not the focus here. This vision of a Web application is best supported by a distributed component model such as Web Services to provide for usage of external functionality within the Web application.

In addition, the consistent specification of interfaces and repository entries in XML vocabularies, as in Web Services' WSDL and UDDI, facilitates integration in an application generation process. The interface definitions can use a common vocabulary and schemes with the domain and context ontologies (that are also based on XML); when this is the case, service parameter and results can be immediately interpreted in the application. From a technical perspective, the XML-based publishing system has built-in mechanisms for processing XML definitions, such as a Web Service interface, and furthermore for manipulating service call results, which are also structured as XML.

Service Incorporation

Service incorporation in an application at the top-level, meaning, where and when is which service resource presented, is specified in the same manner as for the other types of resources (see Section 3.4). Therefore, the selection of services can be context-dependent: in the context model, a service can be modeled to be relevant in a certain context, just as any other resource. As for the other resource types, the choice of which services are to be incorporated at what point in the Web application, and where the presence of a service offering is not predetermined but should instead be selected according to context, is a design decision for the Web engineer when designing an application.

Integration of services raises additional challenges to those of static types of Web application resources. Considering existing services that are known at design-time, the question is how context information can be used to enhance the integration; regarding cases where the designer wishes to integrate a service of a certain category, but where a concrete service is not known, the question is how a service can be dynamically found at run-time.

A fully specified service can be referenced by name in the conceptual model, and henceforth stored in OWL with a reference to the complete service specification in WSDL. The following approaches can then be used to achieve incorporation of services in an application:

- As OWL and WSDL are XML formats, appropriate XSLT stylesheet transformations process service definitions and generate an appropriate XSP, filling in Cocoon's built-in SOAP call template. A Web Service is called by integrated Cocoon functionality, allowing to forgo the need for a custom implementation in program code.

- A custom architectural component is defined that can process service resources in a generic way. Such a component accesses application models and the service's WSDL description, and can

use the components for context to assist in adaptation. The component calls the Web Service when required and makes results available to further pipeline processing. Defining a component in this manner is conceptually similar to the *SQL Transformer* example from Section 4.2.4.

Another type of service integration concerns services that are not fully known in advance: in this case, only a name for this service is modeled and consequently stored in OWL. This name can be a category to be looked up in UDDI directories for a corresponding available Web Service. UDDI Green Pages are a registry for Web Services' technical information, and how to interact with them, typically by providing a pointer to a WSDL definition. This service's WSDL can then be retrieved by the system, and processing continues as above.

User Interfaces and Web Services

An integrated Web Service may or may not require interaction (meaning, require user action to be taken before being executed). If use of a service requires interaction, a suitable user interface must be provided. In both cases, the service might return information that needs to be displayed in the user interface.

Given the goal of presenting the end-user with a means of interacting with a Web Service, the most straightforward approach is to develop a custom user interface for the Web Service: if the developer knows which information about the Web Service is to be presented to the user, and what information the user must provide in order to consume the service, then the developer can develop a custom software client component, for instance a Java-Swing GUI or an XHTML form-based application. This type of approach, however, raises typical issues of reuse, cost-effectiveness and quality assurance.

The alternative is to (automatically) generate a user interface for the Web Service. In such generation, a particular challenge is to support the dialog in a user-friendly manner, meaning achieving a suitable interface for each service, while having a generation process of general validity. One approach at such a generation is to provide appropriate XSL templates which can transform an XML representation of the service interface into, for example, an XHTML form or an SVG[4] animation, and to use context information (added by a custom architectural component) to enhance interaction. For example, consider a Web Service that requires n parameters of which only the name and a general type definition (such as the "any" type) is known, but nothing further. An applicable GUI design pattern can be to apply an XHTML form prompting the user for required values. The more elaborate information regarding required parameters is, the more sophisticated the user interface can be, for example, to automatically validate input according to an underlying XML Schema. If parameter types have meaning to the Web application (for example, when the types are defined in the ontology), the context engine can be queried to see if this information is already known from context. If known, the information can be provided as default value in the user interface. Optionally, a service call for which all required parameters can be provided from context can be automatically triggered, requiring no user interaction at all - this is of particular interest in pervasive computing situations. Context information can thus be used to automate dialogs between the end-user and a Web Service, or to achieve some degree of automation. Which service is to be automatically called, and which should always require user confirmation, is a design decision for the Web engineer.

The result of a Web Service call is returned as XML, thus XML-based pattern mechanisms can also be applied to render the result of the call to a service. For example, if an array type is returned within the result, this can be matched to an appropriate XSLT template transforming an array into, for example, an XHTML table.

[4] Scalable Vector Graphics

Furthermore, the result returned by a Web Service can, in some situations, be used as input for another Web Service. For example, one Web Service might provide a list of products in answer to some criteria selection by the user; and another Web Service might provide full details for one specific product (such as description or availability), product identification being shared between both services. To combine such Web Services as user interface elements, finite state machines such as described, for example, in (Kawash, 2004), can define transitions between Web Services, as well as parameters needed for a transition.

Binding Web Services and Context

As illustrated, when integrating an existing Web Service into an application, values for input parameters to the Web Service should be derived from context where possible. In other words, an optimal integration of Web Services and context requires that the interface the Web Service provides can be augmented by context information. After service call completion, context can be used further, for example to guide presentation of the service call's results. If, in addition, the semantics of the information returned by the service are known, the results can constitute a cue for context extrapolation.

However, the goal of optimal binding raises a problem: how does the system know that a parameter X of a service can be determined by a (currently known) context factor? A partial solution is to use shared, predefined context factor catalogs. (Strang et al., 2003a) propose a model of how to catalog "context attributes" for Web Services. Such a cataloging can not be exhaustive, but can provide assistance in some scenarios: external Web Services can use the catalogs for specifying the parameter types, and an application calling the service can provide parameter values according to the sensed context. If this knowledge is specified in ontologies (OWL) just as the domain knowledge is, this allows for direct, run-time mappings in the system.

Ideally, common classifications will emerge. The above-mentioned UDDI stores classification and identification information, for which standardization efforts exist: for example, UNSPC for Product and Service Classification. For the context factors, examples of standardization efforts are provided in Section 3.4. If a Web Service supports the use of these classifications for its parameters, mapping of context factors onto parameter values can be immediate.

To further augment quality of service incorporation, introspection can be used. Consider for example a service "productSearch" of an online store for clothes, requiring a "clothesType" and a "size" parameter. Presume the user has explicitly selected the type "shoe", by navigation or selection in a list. Introspection within the user information could mean to look if an attribute "shoeSize" exists, and use it if available. If not, the system can look for "size" as attribute, and make a plausibility check that the value is actually the shoe size, and not something else. If ambiguous, this parameter is marked as uncertain, and before the service is called, the user would be asked to confirm this parameter. This example illustrates how, by using context, demands on the user can be reduced for the use of services, but, as a general matter, not avoided entirely.

4.3.4 Navigation and Patterns for Presentation

Ontology-based navigation is an approach to supporting navigation that is well-suited for integration with context handling. Navigation corresponds to concepts in the domain ontology, or instances of these concepts. When a user has navigated somewhere in the application, the system thus knows the currently active concept. The system can then determine from the model which other factors (further concepts, resources, context factors) are associated.

For rendering of user interfaces, we propose to apply patterns to the ontology-based navigation and to the dynamically generated application parts. This notion of patterns for user interfaces is to be understood as executable patterns, meaning patterns that are specified in such a way that a system can interpret and apply them. This is similar to the notion of design patterns in (Bosch, 1998), where such patterns are specified in a formal language and then translated to C++ code, such that the code can be used within a C++ program. Here, we propose to specify patterns in XML in such a way that they can be used for Web user interface generation.

Navigating to a node results in generation of one or several pages in the application. For example, a document might be rendered in one page if the device used is a desktop browser, and in several pages ("cards") if the device is a mobile (WAP) browser. This is generally true, independently of a specific technical constraint. Logical navigation to a node in the ontology can result in several steps of concrete navigation, for which a pattern is applied: for example, a tabbed interface for a larger piece of information, or a wizard for a service requiring at least one additional step of interaction to confirm parameter values. To describe this requirement, one solution is to model placeholders for navigation nodes, along with a list of applicable user interface patterns: content of Web pages is thus specified on a more abstract level than the immediate association of documents to navigation nodes, as would typically be the case in a CMS approach (see Section 2.2.2). In this work's notion of Web applications, content can furthermore be provided by service calls (for retrieving dynamic information) or represent interfaces allowing the user to call a service. Interaction with services raises particular issues that are addressed in Section 4.3.3.

Generally speaking, to create a graphical user interface (GUI) for a Web application, it must be specified which GUI patterns are appropriate for which resource types, in which context. On the one hand, information or data structures provided by a resource must be rendered, and on the other hand, user interaction with the resource must be enabled, when applicable. Beyond the rendering of individual resources, rendering and positioning of the resources in relation to the other resources must be supported by the system. Portal approaches (see Section 2.2.1) provide ideas in this direction; for instance the Cocoon portal module allows to specify the layout of portal elements in patterns (represented in XML), for example, in rows and columns. For general-purpose Web applications, such an approach is adequate for rapid prototyping, but will presumably need to be refined in order to empower the designer to define more precisely desired look and feel. For example, (Grundy and Yang, 2003) proposes to allow designers to define the layout according to the principles defined by Java's *GridBagLayout*, and then translate this specification via XSLT into a form appropriate to the current device. For Web browsers, the design is translated into embedded HTML tables; for mobile browsers, into a set of WML cards. An alternative approach is to define layout constraints between the user interface parts, meaning that parts are to be positioned relative to the position of the other parts; see, for example, (Vandervelpen and Coninx, 2004).

To ultimately render a GUI in the XML- and Cocoon-based approach, pipeline fragments first generate an XML representation of contents of each Web application part appropriate for the current user request. The contents are then aggregated in a Cocoon pipeline. Transformer components successively transform the format as discussed in Section 4.3.1, to account for context adaptation and service calls. Once all required information was successfully generated, the GUI rendering phase is started, and also uses XML transformations. The GUI can be structured into several pieces, for example to explicitly separate between a navigation and a content area. Whatever the actual structuring is, each structure is rendered with a pattern responsible for generating a user interface fragment (such as an XHTML fragment) for this structure only. User interface fragments are then collated to produce a Web page.

GUI Rendering with XSL

The notion of executable GUI patterns for Web applications as described above can be realized with XSL technology.

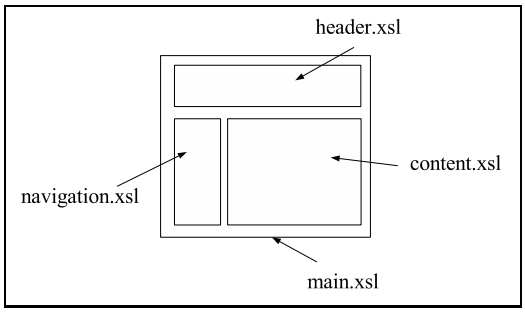

Figure 4.7: Example of top-level user interface structures and stylesheets

Figure 4.7 illustrates a classic structuring into navigation, content, and header, each rendered by individual stylesheets which are then aggregated in a general purpose stylesheet. The way the individual structures are positioned is in fact not fixed, but can be determined, for example, by CSS entries, which can define absolute or relative positioning.

Application of XSL stylesheets for each structure can be recursive. For example, if a pattern "tree" is to be applied to the navigation structure, each node in the tree can be rendered by an XSLT template that in turn applies templates to the node's sub-nodes. Likewise, the content structure is a container for potentially several pieces of content; for each piece an appropriate XSLT template can be applied, either related to the type of content, or according to pattern information. Which pattern is to be used can be specified in the model, or be dynamically determined by an architectural component, potentially with assistance from the context system.

```
<!--
   A template for rendering a response from a service call,
   according to 'table' pattern
  -->
<xsl:template match="content:serviceResponse[@content:pattern='table']">
   <div>
      <table>
         <xsl:for-each select="content:entry">
            <tr>
               <xsl:for-each select="content:value">
                  <td>
                     <xsl:value-of select="."/>
                  </td>
               </xsl:for-each>
            </tr>
         </xsl:for-each>
      </table>
   </div>
</xsl:template>
```

Listing 4.2: GUI pattern and XSLT templating example

Template matching mechanisms can use XPath statements to determine which template is applicable in a given situation. In the example of Listing 4.2, presume a piece of content generated by the pipeline (consequently to a service call), and furthermore a transformer component which determines

that the pattern `table` is appropriate in this situation to render this piece of content. The transformer dynamically augments the content description with a `pattern` attribute, thus allowing for straightforward usage of XSLT template matching mechanisms to choose the appropriate GUI transformation instructions.

4.4 Summary

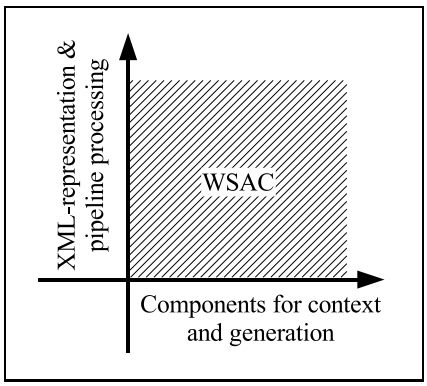

Figure 4.8: Principles for a SoC-based Web software architecture for context (WSAC)

This chapter has described a *Web software architecture for context* that supports both the dynamic generation of Web applications and adaptation functionality. Two orthogonal approaches assist in realizing the Web software architecture for context (see Figure 4.8). On the one hand, an information representation and XML processing concept, called pipelining, defines how user requests are handled and how the application is dynamically generated. On the other hand, individual components, designed according to the separation of concerns principle, implement sets of functionality supporting a Web application, where each component provides a clearly determined feature, either supporting context-awareness, adaptation, or achieving a step of the generation process.

In the following chapter, it is shown how the architectural approach and corresponding techniques described in this chapter can be implemented in order to support adaptive Web software, by means of a framework for context including components for context. The usage of the Web application models of Chapter 3 in the architecture is described in that framework and demonstrated in a corresponding case study.

For completeness, it should be pointed out that basing a software architecture on XML also carries risks, notably regarding software development and quality, as discussed in (Dumke et al., 2003):

- the existence of several layers of interpretation (such as XML and XSL) can result in error inheritance, thus making debugging more difficult. On the other hand, the layers permit a separation of concerns, facilitating debugging if the layer is correctly identified.
- as several layers are involved, there is a coupling of layers that must be understood by the developers, and also by system maintainers.
- several standardization attempts currently exist to define common usages of XML, that might simplify and clarify system architectures in the future, but these standardization efforts are not yet mature. For the time being, compatibility and portability between software solutions are not guaranteed - an agreement on XML as a common syntax is not sufficient for those purposes.

- the system is based on interpreting XML contents and content descriptions, and thus has the advantage of high variability in generation. The risk is that the generation cycle cascades out of control in the run-time system.

Chapter 5

The CATWALK Context Framework

This chapter describes CATWALK[1], our software framework for context. This framework supports systematic usage of adaptation to context in Web applications, by providing run-time components that are steered by the artifacts resulting from modeling as explained in Chapter 3, and by implementing the architecture described in Chapter 4.

Section 5.1 gives an architectural overview of the framework; Sections 5.2 and 5.3 detail the custom components for context and adaptation processing, and how they are integrated in the architecture. Section 5.4 explains how context can be simulated for prototyping or evaluation purposes. Section 5.5 provides the proof of concept by means of a case study, in which a prototypical application is described in terms of application models only, and which shows examples of adaptation effects in the application generated by the framework. Finally, Section 5.6 contains a discussion of functional, design, and technical aspects of the framework.

5.1 Architectural Overview

The framework consists of components for context-awareness, adaptation, and application generation; and furthermore of XSLT stylesheets and pipeline fragments that define processing steps integrating these components. An architectural overview and its relation to the application models is given in Figure 5.1.

The modeling methodology as described in Section 3.5 produces artifacts that are made available via a model repository. This is represented by the upper half of Figure 5.1. The model artifacts are defined in XML (respectively, XML-based dialects such as OWL). Access to these artifacts is encapsulated by the definition of a Cocoon pseudo-protocol: this means that each artifact can be accessed via a special URI prefix (in this case, the prefix `wise://`) within a Cocoon pipeline or processing component. This abstraction allows to delegate physical access to the models to a specific implementation: the default implementation reads from the filesystem, whereas an alternative implementation might read the models from a network service, for example a KAON RDF server (Oberle et al., 2005), to better support decentralized development.

The lower half of Figure 5.1 presents the main components in the CATWALK architecture, and their sequencing within the Cocoon pipeline. Arrows with a white background indicate this flow: the client request is matched in a Cocoon pipeline, and processing continues through components responsible for application generation, ultimately resulting in the response to the client, for example,

[1] Context-aware Adaptation through Transformations for Web Applications Leveraging Knowledge

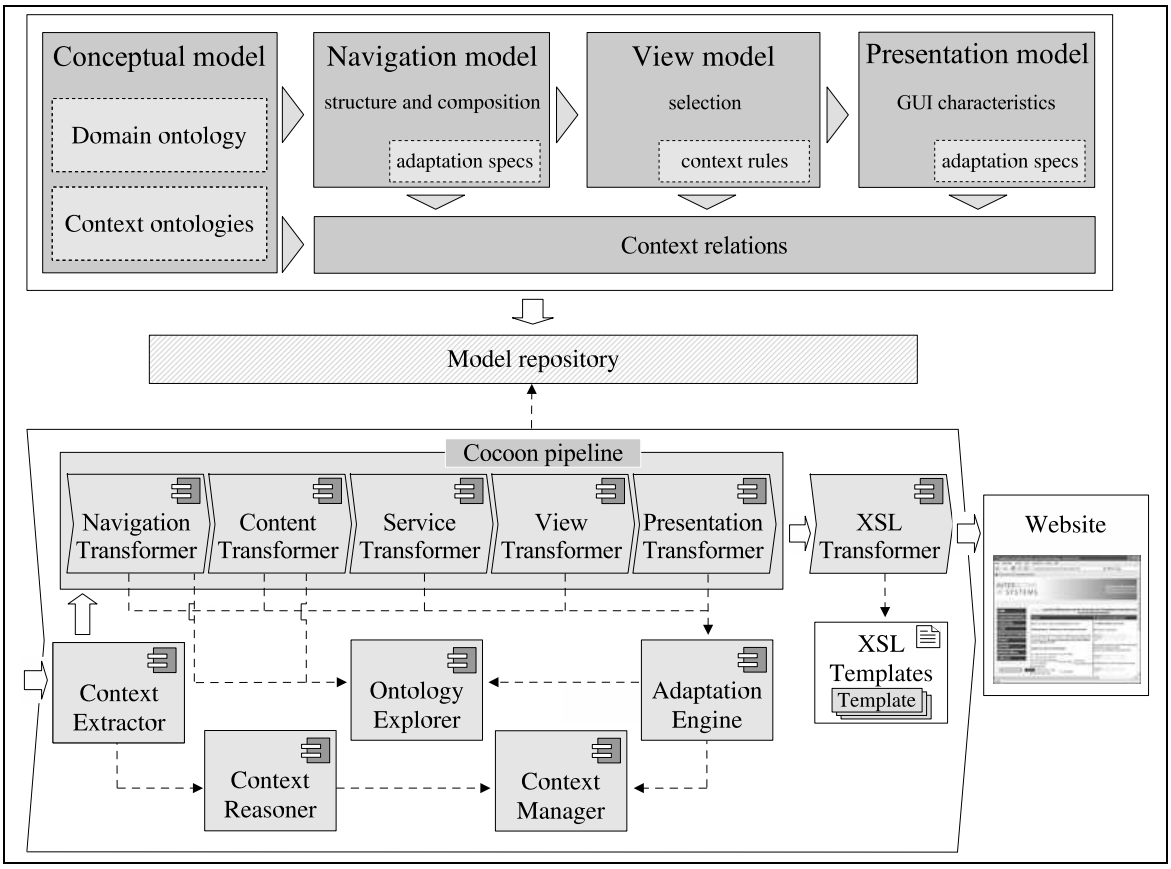

Figure 5.1: Context framework component architecture

as a page of a Website. Arrows with a dotted line originating from a component indicate that this component calls another component or uses other system elements for its processing.

Each component in CATWALK implements a specific concern, in the sense of the *separation of concerns* architectural design principle (see Section 4.3). A component in this sense is implemented by one or more Java classes and may use additional artifacts (such as XSLT stylesheets for XML transformation). The Java classes realizing the component interface are furthermore extensions of existing Cocoon component types; this means that the new components in effect augment the Cocoon framework, and may thus be used within regular Cocoon processing. Pipeline processing fragments describe how the components are integrated via Cocoon pipelining mechanisms, and delegate to XSLT stylesheets for intermediate XML transformations.

The framework's components can be assigned to one of two categories. The first category consists of components for context that implement the *Context System* subsystem as described in Figure 4.2, and thus provide core functionality for context-awareness and reasoning; these are described in detail in Section 5.2. The second category consists of components responsible for adaptive Web application generation; these implement the *Application Generator* subsystem shown in the same Figure. These components are detailed in Section 5.3.

5.2 Components for Context

This section discusses how CATWALK provides mechanisms for context-awareness and adaptation. We discuss components for handling context representation and sensing, followed by a component supporting adaptation functionality.

5.2.1 Context Manager

The *ContextManager* component is responsible for handling access to context state; that is, this component allows other components to obtain information about context (present and past), and to store information about context. The term "context" is used as defined in Section 3.3.1. The "context state" is a projection of this definition of context onto n activated context factors.

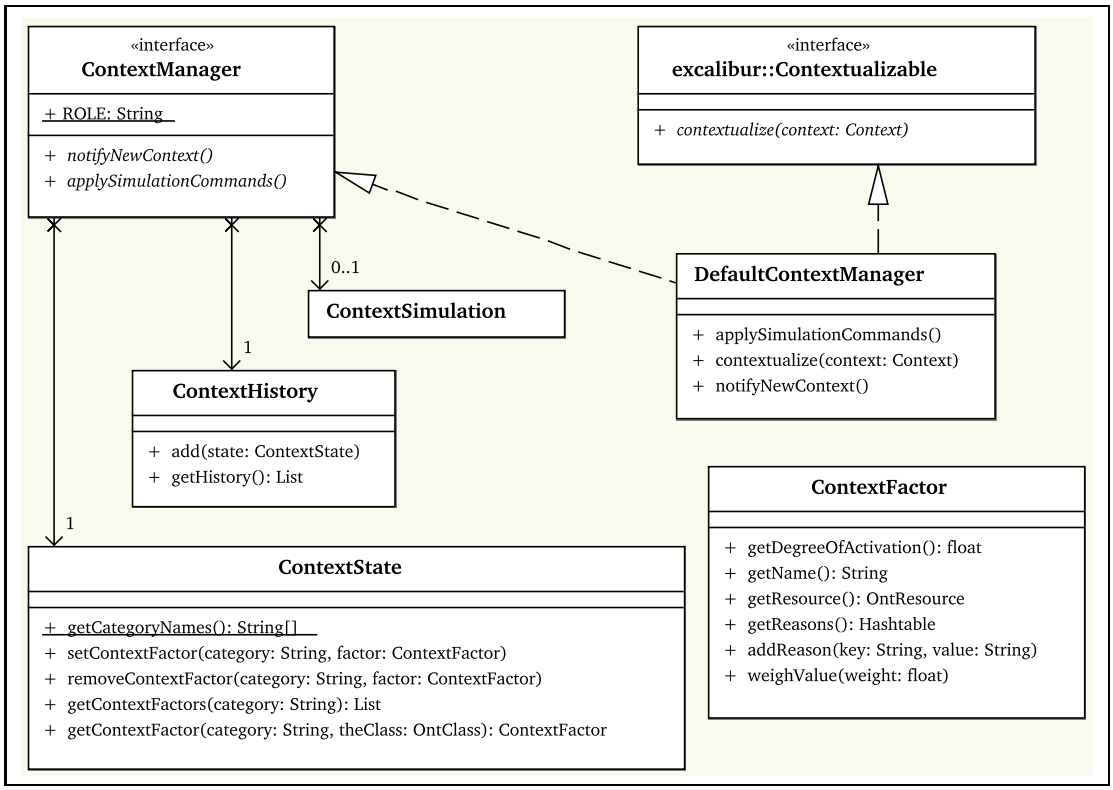

Figure 5.2: *ContextManager* class diagram

Figure 5.2 provides a class diagram for this component. The component is registered as an Excalibur service; it can thus be looked up using the interface's *role* (whose name is defined by the constant `ContextManager.ROLE`). Components requiring knowledge about context state can retrieve a `ContextState` instance via a *ContextManager*, as expressed by the association in the class diagram. Likewise, a `ContextHistory` object is accessible via a *ContextManager*. The `ContextState` in turn is structured into n categories of context. The interfaces assume a categorization of context factors, however which actual categories of context exist is unimportant for this component; the definition of categories of context can be altered in a straightforward manner. For each category, a set of `ContextFactor` instances is associated. Such a `ContextFactor` has a degree of activation, and may contain a reference to an entry of an ontology. Furthermore, a `ContextFactor` stores a list of

"reasons", indicating why this factor was activated; the degree of activation can be altered in relation to the current degree by using the `weighValue` method.

A component wishing to add or update a context factor in context state should first notify a *ContextManager*, access its `ContextState` instance and call the `ContextState`'s `setContextFactor` method. The following section (5.2.2) describes mechanisms for determining context factors. In addition to regular context state handling, the *ContextManager* can apply so-called simulation commands to override the state. Context simulation is described in Section 5.4.

A default implementation for the *ContextManager*, called `DefaultContextManager`, is provided. In this implementation, context storage is handled via the current user session of the servlet engine. Another implementation might, for example, handle persistent storage of past context in a database. To specify which implementation is to be used for this component, the class name is given in the role configuration file (`catwalk-roles.xml`).

5.2.2 Context Sensing

The *Context Sensing* architectural component, as introduced in the composite structure diagram of Figure 4.2, is realized in CATWALK via two actual software components: the *ContextExtractor* and the *ContextReasoner*. The *ContextExtractor* is responsible for determining information about context present in the user's current request; that is, for translating any parameters of the request (present in the URL and in the protocol) into the terminology and granularity used in the application models. The *ContextExtractor* then transmits its current information about context to a *ContextReasoner*. This component in turn is responsible for determining degree of activation of the domain and context ontology elements, and for updating context state, via a *ContextManager*.

Context Extractor

The *ContextExtractor* component is responsible for extracting context information from the user's current request. The component must therefore be used upon each user request, and called before any further pipeline processing (see also Figure 5.1). The component delegates to specific implementations for each category of context, allowing to use different mechanisms for each category.

```
<!-- Extract from catwalk.roles, component declaration file -->
<role name="info.interactivesystems.catwalk.components.roles.ContextExtractorSelector"
    shorthand="request-context-extractors"
    default-class="org.apache.cocoon.components.ExtendedComponentSelector"/>

<!-- Extract from cocoon-xconf.xml, component configuration file -->
<request-context-extractors>
    <component-instance name="location" logger="catwalk.extractor"
        class="info.interactivesystems.catwalk.components.DefaultLocationContextExtractor"/>
    <component-instance name="device" logger="catwalk.extractor"
        class="info.interactivesystems.catwalk.components.DefaultDeviceContextExtractor">
        <patterns>
            <item>
                <value>desktop</value>
                <pattern>Mozilla</pattern>
                <pattern>MSIE</pattern>
            <!-- etc. -->
```

Listing 5.1: Context extractor configuration

The *ContextExtractor* component is specified as an Excalibur service realized as a collection of implementations, each realizing a common interface, but distinguishable by a "name" (the name of the context category is used). This setup allows for using Cocoon configuration mechanisms to specify which categories of context are to be processed, and which implementation accomplishes this, as illustrated in Listing 5.1. Default implementations are provided for each category of context; the class diagram is shown in Appendix D, Figure D.1. The component configuration file can contain additional configuration information for individual implementations. In the example of the extractor for the "device" category, sets of patterns are provided; when a request parameter is matched by a pattern, the corresponding value is automatically used as context factor. Such a configuration can thus reduce the need for custom programming (this mechanism is similar to Cocoon's built-in "browser selector" component).

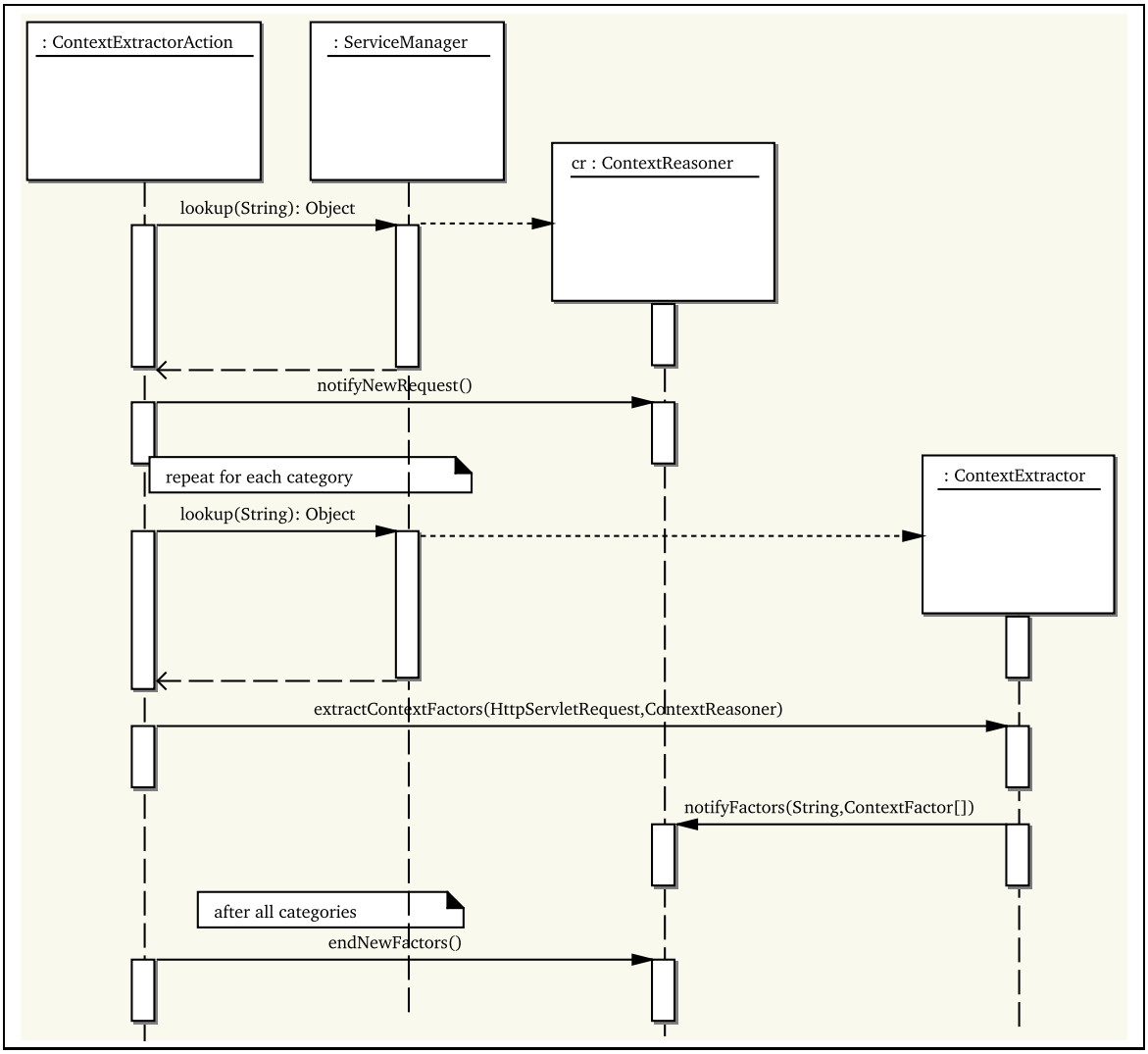

Figure 5.3: Request context extraction and notification sequence diagram

The order of request processing and context notification is illustrated in the sequence diagram of Figure 5.3. The component is hooked into pipeline processing by the `ContextExtractorAction` class, which is an *Action* (in the Cocoon sense) configured to be executed upon entry of the main

pipeline used for processing in CATWALK; this *Action* is thus used for each new user request. The *Action* first queries the framework for an implementation of the *ContextReasoner* component, and then for an implementation of *ContextExtractor* for each known category of context. Each implementation is provided with the current request and a reference to the *ContextReasoner*, which it notifies regarding the recognized factors.

The base class `AbstractContextExtractor` is used to simplify implementation of individual context extractors (see Appendix D, Figure D.1). This class handles notification of results and pattern matching according to component configuration. Applications requiring other context extraction techniques than those provided by default may subclass `AbstractContextExtractor`, and register their class name as implementation within `cocoon-xconf.xml`, the component configuration file.

Context Reasoner

As discussed in Chapter 4, context information is gathered from information directly available in a user request or provided by external sensors, but knowledge about context can also be obtained by using additional sources, such as the relation of recognized factors to other elements in an ontology (see Chapter 3), or past context information.

The *ContextReasoner* component is notified of context factors present in a user's request, and is responsible for determining current degree of activation of all domain and context ontology elements. A *ContextReasoner* is thus responsible for computing knowledge about context in the largest sense.

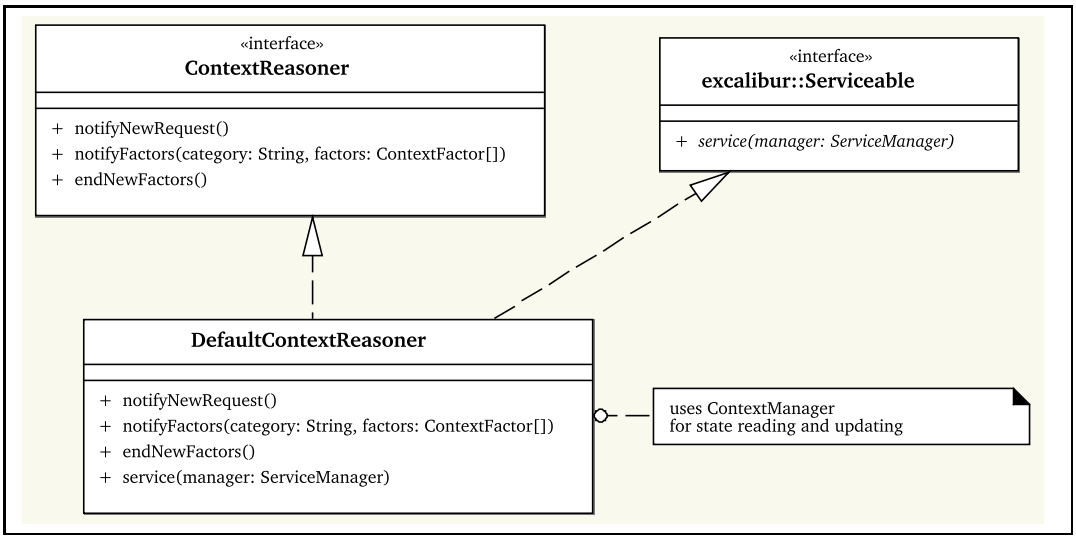

Figure 5.4: *ContextReasoner* component class diagram

Figure 5.4 describes the interface for this component. The *ContextReasoner* can use the *ContextManager* for reading previous context, meaning the context state prior to this request, and any past states. Once it has determined the new context state, the *ContextReasoner* notifies the *ContextManager* to store this state.

To compute current context, implementations can use different techniques, such as spreading activation as used in (Rocha et al., 2004), Bayesian networks (Tipping, 2001), or heuristics based on analyzing past context activation degrees in addition to currently recognized factors, such as case-based reasoning as used in (Mikalsen and Kofod-Petersen, 2004). CATWALK provides a default implementation, the `DefaultContextReasoner`, that uses present and past user navigation to compute domain

context, and sets factors in the other categories of context according to the values computed by the *ContextExtractor* for each category.

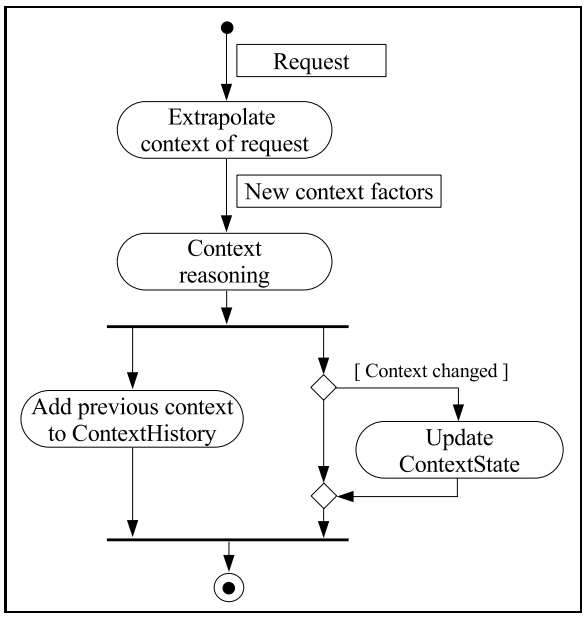

Figure 5.5: Context sensing activity regarding context state

Context sensing activity (as realized by *ContextExtractor* and *ContextReasoner*) with regard to context state is summarized in Figure 5.5. The user request is processed, as described above, by n implementations for extraction; these implementations notify the *ContextReasoner* of new context factors. The *ContextReasoner* in turn triggers changes in context state, consequently to context interpretation. Depending on implementation of the reasoning, different points in time are conceivable for state updating. If reasoning is fully contained within each category of context (i.e., only current and past factors recognized within this category are used for reasoning), results for each category can be immediately transmitted to the *ContextManager*. If, however, results from reasoning on one category may influence results for another category, the *ContextReasoner* may only define the new context state once all categories have been processed. To support such a process, the *ContextReasoner* interface defines the `endNewFactors` method, which is called by the `ContextExtractorAction` as indicated in Figure 5.3.

The state of context must be checked upon each request, but state itself need not change. State is considered changed when at least one of the context factors (as computed by context sensing), in at least one of the categories, is changed. For instance, if an intranet user has a new network address consequently to a temporary network outage, but context sensing still recognizes location as "intranet", a context factor "network location" would not change.

History of context, however, changes with each request. Even when context state is unchanged between two requests, this fact is significant for the history, as it provides a measure of time the user has stayed in that context. This information can be used by a reasoner implementation using context history.

Once all processing of the components responsible for context sensing is complete, subsequent pipeline steps have access to the updated context state via the *ContextManager*.

5.2.3 Model Representation and Access

Before proceeding to explain adaptation and application generation components, we need to describe how application models, as described in Section 3.4, are represented and accessed in the framework.

The `ContextModel` in CATWALK is an object-oriented, runtime representation of those model artifacts pertaining to context that have been produced by the modeling activity. This representation is not to be confused with the *context state* (see Section 5.2.1), that in turns represents the current context of a user session. As described in Section 3.5.2, the context model is defined in ontologies, using OWL. The JENA toolkit (Carroll et al., 2004) provides a Java API and implementation for working with such ontologies. CATWALK uses JENA for reading the ontologies from a given source and for using Java facilities to traverse the ontologies and search for specific statements.

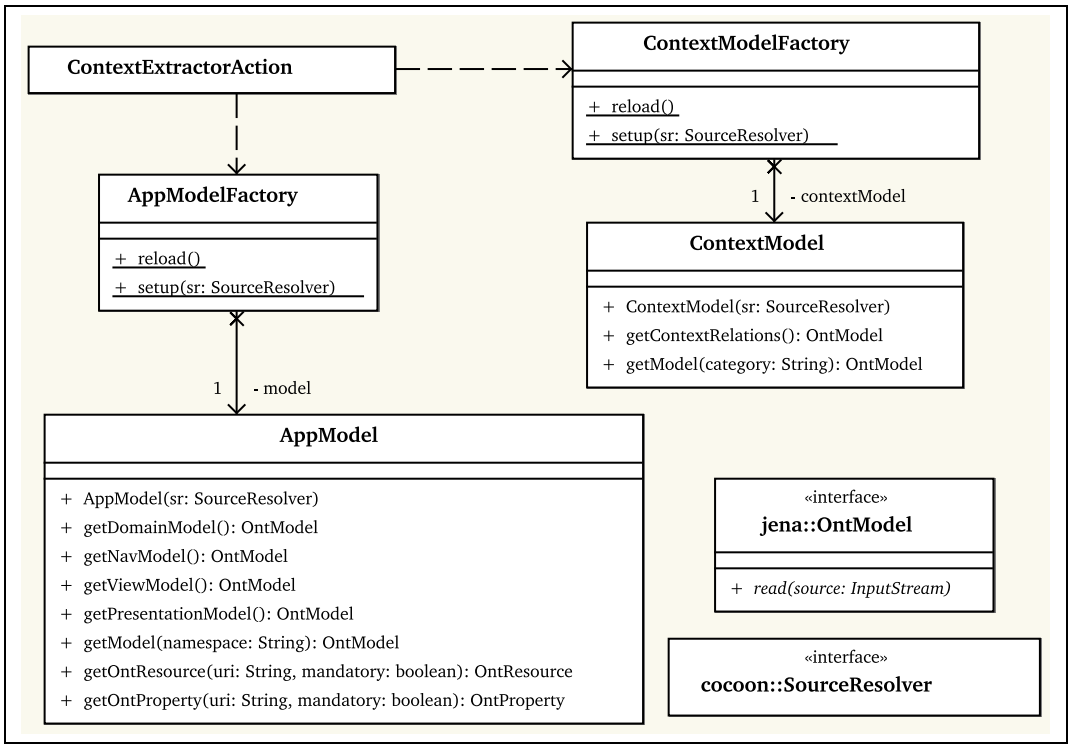

Figure 5.6: Model representation class diagram

Figure 5.6 provides a class diagram for model representation. The `ContextModelFactory` provides a single (static) access to a `ContextModel`: for efficiency, the servlet container reads the ontologies into memory only once, and this instance is shared by all requests. Other components thus access an instance of `ContextModel` by navigating via the `ContextModelFactory`. The `ContextModel`, in turn, encapsulates the individual context ontologies and the modeled context relations.

The application sub-models (domain, navigation, view and presentation, as presented in Section 3.4) are accessed similarly to the context model. The sub-models are encapsulated in an instance of `AppModel` that is accessible via the `AppModelFactory`.

The factories are responsible for creating an instance of the model representations upon first access. The factories must therefore have information about how to read the ontology sources. As introduced in Section 5.1, access to ontology artifacts is encapsulated by a `wise://` pseudo-protocol within the Cocoon pipeline; consequently a Cocoon sitemap component must initialize

the factories with a mechanism to access these sources. This mechanism is handled via Cocoon's `SourceResolver` interface, a sub-interface of Excalibur's corresponding interface used by sitemap components. In CATWALK, the `ContextExtractorAction` (see Section 5.2.2) is used for this initialization. The sequence of processing concerning the `ContextModelFactory` is shown in Appendix D, Figure D.2; the `AppModelFactory` is initialized in the same manner.

The `ContextExtractorAction` is configured, in the Cocoon sitemap, with URI information defining how the ontology artifacts are found (see Listing 5.2); these parameters are passed to the factory `setup` methods and henceforth to model constructors (these parameters are not shown in the class diagram of Figure 5.6 in order to reduce the diagram to essentials).

```
<map:act type="contextsensor">
   <map:parameter name="base-uri" value="wise://model-repository/"/>
   <map:parameter name="crelations-uri" value="contextrelations"/>
   <map:parameter name="uri-suffix" value=".owl"/>
   <!-- ... -->
```

Listing 5.2: Sitemap configuration for model access

5.2.4 Ontology Exploration

The *OntologyExplorer* component allows to retrieve items from an ontology according to a lookup specification such as defined in Section 3.5.3.

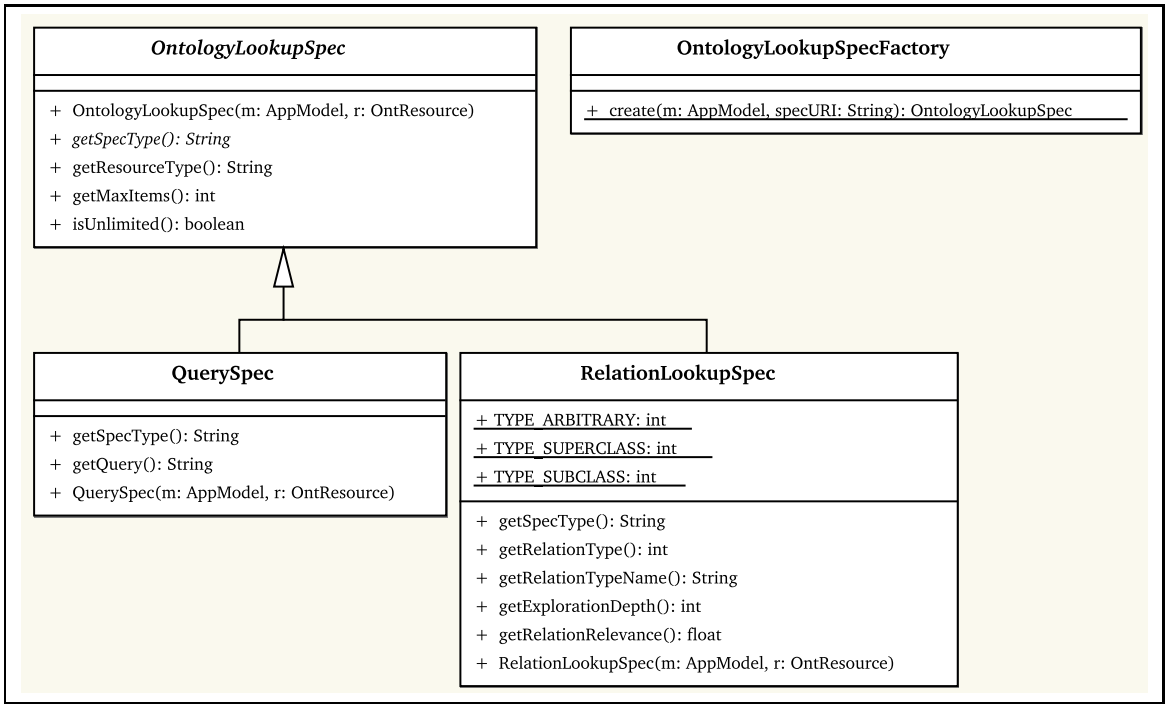

Figure 5.7: Ontology lookup class diagram

The framework provides an object-oriented representation of lookup specifications, see Figure 5.7. A lookup specification can be of two types, `RelationLookupSpec` or `QuerySpec`. The abstract type `OntologyLookupSpec` encapsulates usage of all lookup specifications. To create an instance of

the appropriate type, the `OntologyLookupSpecFactory` should be used, and provided with the URI to the specification's entry in the application model.

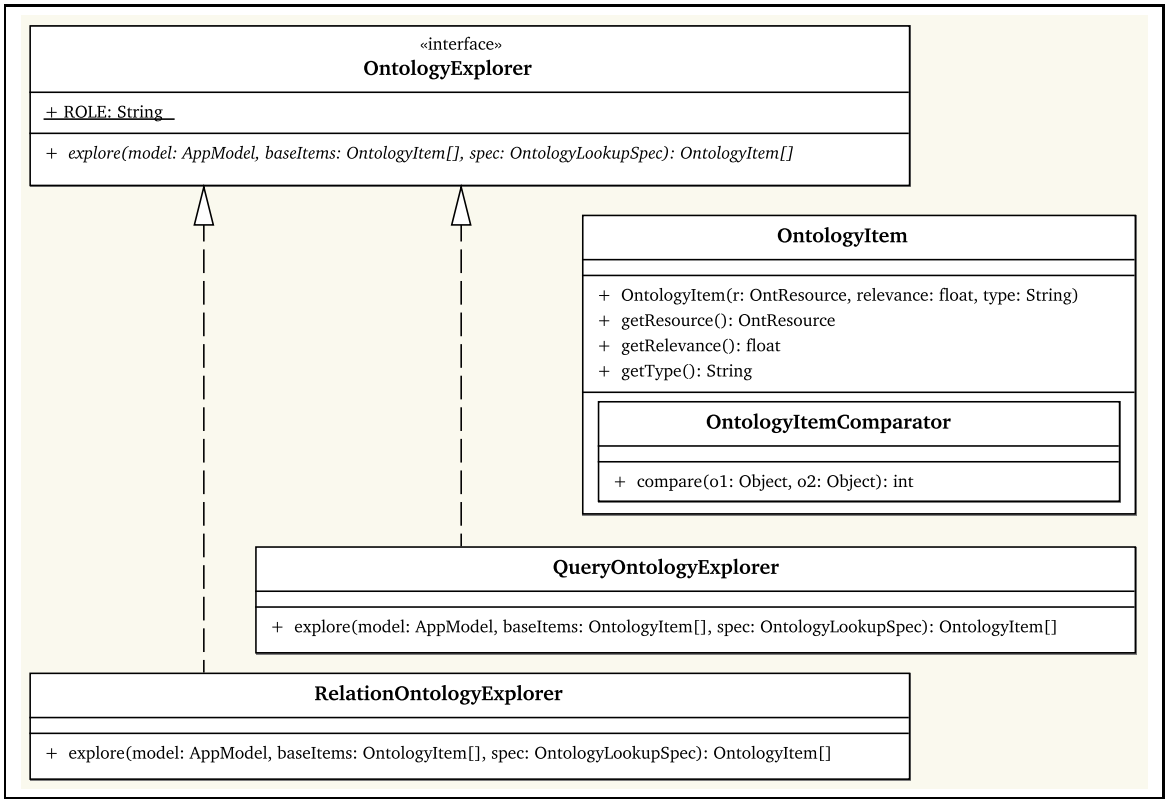

Figure 5.8: Ontology exploration class diagram

The *OntologyExplorer* component role (see Figure 5.8) defines an `explore` method, whose functionality is to search an ontology for items according to a strategy. The search starts with one or more instances of `OntologyItem`, is guided by a lookup specification, and results in computation of further `OntologyItems`. An `OntologyItem` is an object-oriented representation of an element of an ontology, and stores a relevance value for this item as determined by the search algorithm. This allows, for example, to differentiate in relevance according to the depth of relation following in a search operation, and therefore to sort items according to relevance.

As introduced in Figure 5.8, CATWALK provides two implementations for the *OntologyExplorer* component, explained in the following.

The `RelationOntologyExplorer` uses a `RelationLookupSpec`. This ontology explorer depends on the type of relation specified. If the relation type is one of the predefined types (sub-class or super-class), specific ontological knowledge can be used for lookup. For example, if the relation type is "superclass", the initial items' classes are used (taking into account whether an item is already a class or is an instance), and all super-classes retrieved, up to the hierarchy level defined in the exploration properties. Each new level carries a potential relevance loss as specified, again, in the exploration properties. If the relation type is instead arbitrary, the initial items are checked for relations named precisely as defined in the `RelationLookupSpec`.

The `QueryOntologyExplorer` is the second existing implementation of the *OntologyExplorer* component. This implementation interprets the query string defined in the `QuerySpec` as a SPARQL

(W3C, 2005) query, submits the query for execution via the JENA toolkit, and looks up returned URIs in the model in order to convert the response to instances of `OntologyItem`.

To use an *OntologyExplorer*, a lookup specification object must first be created. Subsequently, the framework can be queried for a corresponding implementation of an *OntologyExplorer*. The method `getSpecType` of the `OntologyLookupSpec` is used to provide a hint for the component selector (the component selection mechanism is similar to the one used for the *ContextExtractor*, see Section 5.2.2).

5.2.5 Adaptation to Context

The *AdaptationEngine* component provides adaptation services according to the context. The component role defines three sorts of adaptation services; see Figure 5.9. One such service is the computation of context-relevant items. Furthermore, the component supports context-sensitive service incorporation, by computing default values for service parameters according to context. Finally, the *AdaptationEngine* can determine whether a certain context is currently given.

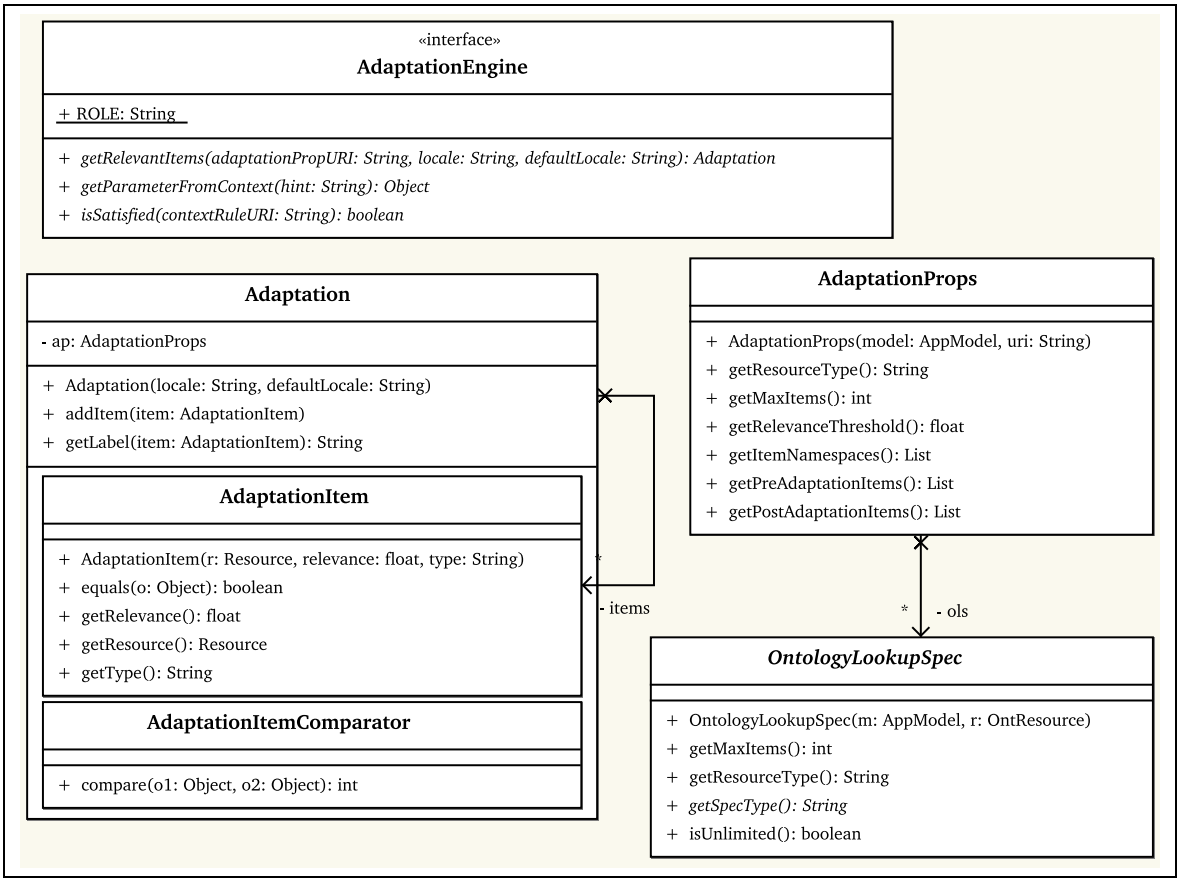

Figure 5.9: *AdaptationEngine* interface

The first functionality of an *AdaptationEngine* is expressed by the `getRelevantItems` method. This method is parameterized with an URI referring to an adaptation specification defined in the model as outlined in Section 3.5.4. This URI can be looked up in the application models via an `AppModel` instance as described above, and used to construct an instance of `AdaptationProps`. This

class in turn provides an object-oriented representation of an adaptation specification in a model. In particular, an instance of `AdaptationProps` can contain any number of `OntologyLookupSpec` instances (see Section 5.2.4). The *AdaptationEngine* responds to a call to `getRelevantItems` with an instance of `Adaptation`, which is in essence a set of `AdaptationItems`. The set is sorted according to context relevance (the sorting is implemented by the `AdaptationItemComparator`). Each `AdaptationItem` may contain a `Resource`, which refers to an entry in an ontology.

The second functionality of an *AdaptationEngine* is defined by the `getParameterFromContext` method. This computation may use service parameter hints from the model, as introduced in Section 3.5.1.

Finally, the *AdaptationEngine* provides the `isSatisfied` method, that determines whether a certain condition involving context factors is currently satisfied. This method is parameterized with a URI referring to a context rule such as defined in Section 3.5.4.

Implementing an *AdaptationEngine*

The *AdaptationEngine* component is defined as an Excalibur service, therefore an implementation can be configured and retrieved using the previously described Excalibur lookup mechanisms. CATWALK provides the `DefaultAdaptor` as a default implementation of this component, discussed in the following.

Computation of Context-relevant Items: The `DefaultAdaptor` determines context-relevant resources in two passes.

First, items that are immediately context-relevant are determined by analyzing the modeled context relations. Each context relation from the model is inspected, and for each relation the number of currently relevant nodes is computed. A node is relevant if the set of active context factors contains an element with a reference to this node. If this context relation has at least one active item, the relevance of the other items is computed according to the relation's influence value factored by the number of items active in this relation. If the relevance exceeds the threshold defined in the adaptation properties, this context relation provides candidates for inclusion in the context-sensitive resource set. Each candidate is then verified according to the remaining adaptation properties: the candidate must belong to one of the desired ontologies (if specified), and be usable as one of the desired resource types. If so, the candidate is added to the result set. If the candidate was already present in the set, the candidate's relevance is set to the maximum of the two values.

In the second pass, the ontology lookup strategies defined in the adaptation specification are pursued, for each lookup specification strategy and each relevant item found thus far. To perform the ontology lookup, an instance of `OntologyLookupSpec` is created, and the framework is queried for an implementation of the *OntologyExplorer* component registered for the corresponding lookup type (see Section 5.2.4). The `explore` method of this implementation is then called to retrieve further ontology items.

A search for context-relevant items must return the n most relevant resources; however, we can not know which resources are most relevant before having examined all candidates, therefore search passes and ontology explorations must be completed regardless of the number of items found thus far. Once all candidates have been evaluated, the n with highest relevance are returned.

Parameter Determination: For parameter determination, the "hint" provided in the model (see Section 3.5.1) is used. The `DefaultAdaptor` first checks if the hint is a URI referring to an entry in one of the ontologies. If so, the list of currently active context factors corresponding to the category

of that ontology is accessed. This is the domain context if the domain ontology is being referred to, or the corresponding context sub-ontology if another category of context is being referenced. The hinted entry is then compared with the list of active factors; a value is matched when either:

- the entry is a class and the set of active context factors contains an element that is an equivalent class or a subclass of the entry;
- the entry is a class and the set of active context factors contains an element that is an instance of that class;
- the entry is an instance and corresponds precisely to an active context factor.

The context factor matching algorithm is of a general nature, whereas the most common case is expected to be that the hint is the name of a class in a context ontology, of which an instance can be found in active context factors, thus matching that instance as return value.

If the hint does not refer to an entry of an ontology, the hint is interpreted to mean the name of an attribute known to the system at run-time. The `DefaultAdaptor` checks attributes of an active instance of "User" within the *User&Role* context ontology - such an instance, when present, corresponds to a logged-in user (as recognized by the `DefaultUserContextExtractor`, see Section 5.2.2). Other implementations of the *AdaptationEngine* might extend this notion to look for values in further categories.

Context Condition Verification: The `DefaultAdaptor` looks up, for each factor referenced in a context rule (see Section 3.5.4), whether the current `ContextState` contains a corresponding `ContextFactor`. If each specified factor is active, the condition is verified. Alternative implementations of an *AdaptationEngine* may offer more elaborate functionality regarding factor combination (for example, as first-order logic statements) and can use the `rule` property for this purpose.

5.3 Components for Application Generation

We now discuss the components responsible for (adaptive) Web application generation, how they are coordinated in pipeline processing, and how they use the components for context discussed in Section 5.2 to integrate context-awareness and adaptation within an application.

The application generation process as introduced in Section 5.1 consists, at its core, of several custom *transformer* components, each requiring adaptation services. Figure 5.10 shows the custom transformer components as subclasses of the `BaseAdaptiveTransformer`. This class is sub-classed by any transformer requiring adaptation. Within the `setup` method, which is called by Cocoon, this class retrieves an instance of the *AdaptationEngine* component. This class also provides access to the current navigation representation, which contains all paths explored by the user thus far. Finally, the `BaseAdaptiveTransformer` encapsulates releasing of the *AdaptationEngine* via the `recycle` method.

5.3.1 Navigation Generation

The first level of application generation consists of determining navigation information appropriate to context. Navigation information in this sense means a representation of nodes that the user can reach from the current position (see Section 3.5.5). This is a structured representation that can then be used for further processing in the application, and not a graphical representation for the end-user.

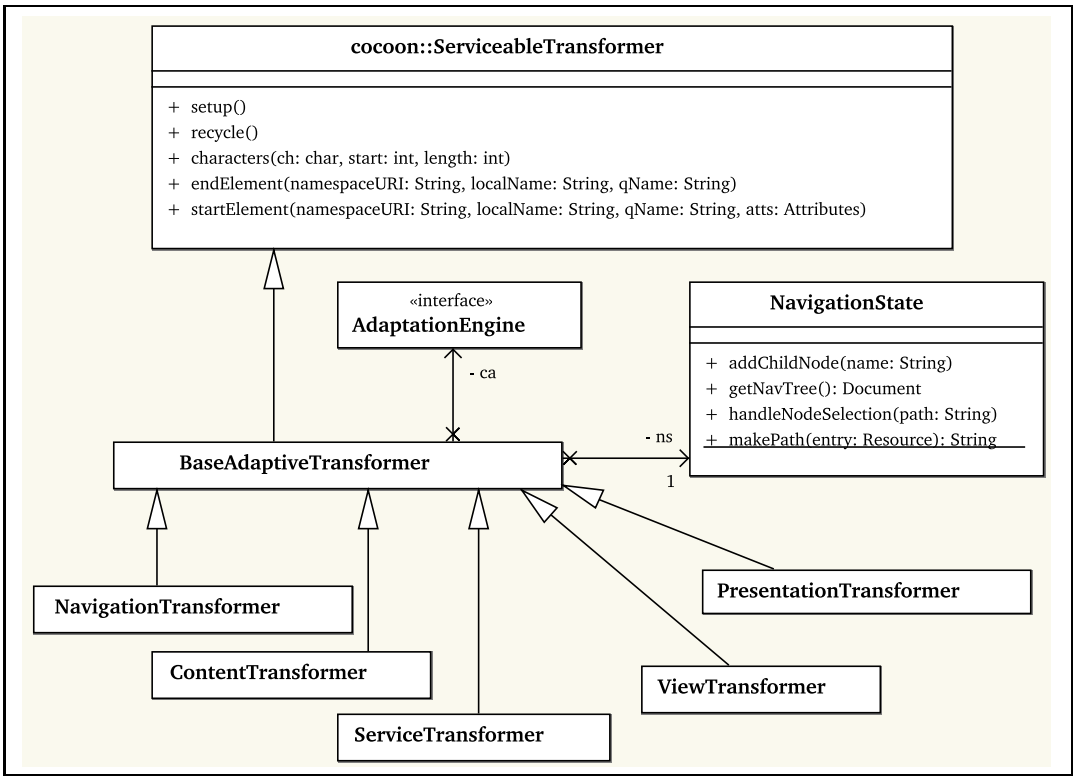

Figure 5.10: Adaptive transformers: class diagram

In classical approaches, navigation generation is determined by the application part that the user is currently visiting; i.e., which node is the user requesting and what information is associated to this node. Some approaches might also take into account nodes previously visited (the navigation history), as context information, to determine which elements to generate next. In our work, the additional requirement is that context is further determined by the other context factors as discussed in Section 3.3.2, and, consequently, navigation generation depends on these factors as well.

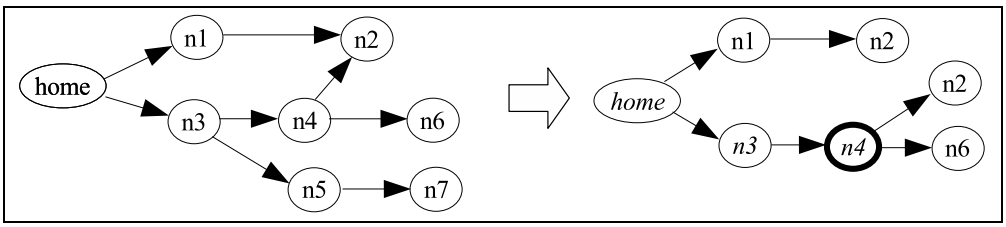

Figure 5.11: Navigation tree example

The process of building a navigation representation from visited nodes is illustrated by the example of Figure 5.11. The left side of the figure illustrates a navigation model, where, for each navigation node, navigation nodes reachable from this node are shown. The same node is reachable via different paths. The path taken is significant, it is part of the navigational context. The navigation model is thus not an acyclic graph and can not be represented as a tree, rather as a collection of uni-directional binary associations. The paths taken, on the other hand, can be represented as a tree, as seen on the right-hand side of Figure 5.11. The paths within the navigation model the user has taken thus far are

stored, and the current path is marked. In the example, presume the user has first chosen node $n1$, then from that node, $n2$. Then, from the $home$ node, the user has chosen $n3$ and $n4$. The current path is $home$ - $n3$ - $n4$ (marked as italics in the figure), and the currently selected node is $n4$ (marked as a bold circle). The navigation tree also shows the reachable nodes from the current node, $n2$ and $n6$.

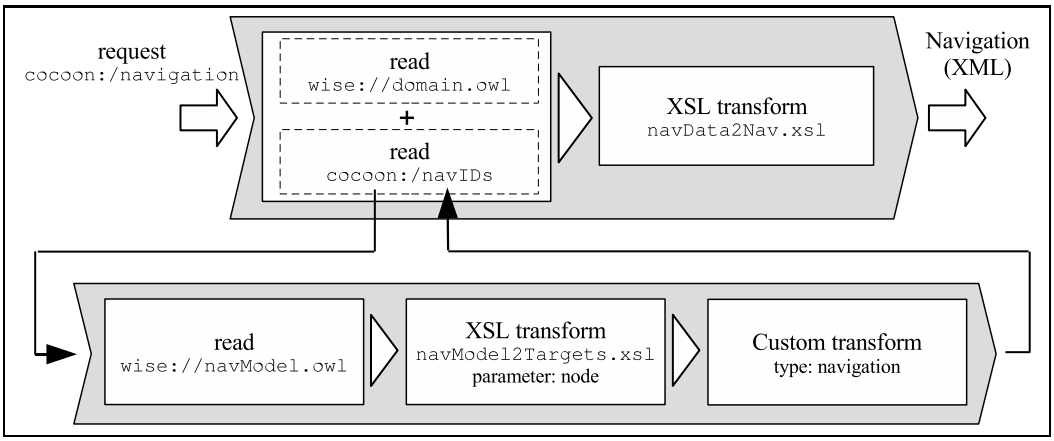

Figure 5.12: Pipeline processing for navigation structure generation

Navigation generation is handled by the *NavigationTransformer* component. This component is implemented with Java classes and XSLT stylesheets and is made available as a pipeline segment identified via the URI `cocoon:/navigation`. The pipeline processing is illustrated in Figure 5.12.

As no standard notation for documenting pipeline processing exists, this dissertation introduces a notation for this purpose. The notation is partially introduced in Section 4.2.4, Figure 4.4, and described in more detail here. A pipeline consists of several processing steps, represented as rectangles. The output of each step is passed on as input to the next. Each processing step is labeled with the type of step on the first line. If the step uses one of the Cocoon standard features (such as an XSL transformation), the second line refers to an URI or to an artifact available to the pipeline. If instead the step requires the use of a custom component, the second line gives the name under which the component is registered in the pipeline (for example, "navigation"). Finally, parameters that are considered essential for understanding the processing step are also listed (for example, "node"). A processing step consisting of aggregation of two or more (XML) sources is represented by a composite rectangle and a $+$. Usage of an URI inside an aggregation can result in a sub-pipeline to be called (for example, `cocoon:/navIDs`). This is represented via arrows from the originating processing step towards this sub-pipeline; the results of the sub-pipeline are then the results of that processing step.

Pipeline processing for navigation structure generation is divided in two phases: the role of the first phase is to extract appropriate navigation nodes for the context, whereas the second phase augments this information with information from the domain ontology (such as node labels). These two phases are detailed in the following.

The first phase of processing is illustrated in the lower half of Figure 5.12 and is marked by the pipeline internal URI `cocoon:/navIDs`. The navigation model, defined as explained in Section 3.5.5, is read and filtered by an XSLT stylesheet for nodes that can be reached from the currently selected node (the targets). These navigation targets are then processed by the Java class named `NavigationTransformer`, registered in the Cocoon sitemap under the name `navigation`. This class implements that functionality of the *NavigationTransformer* which requires specific Java processing. The class acts as a Cocoon transformer; meaning it is both a consumer and producer of XML.

Access to an instance of `NavigationState` is provided via the superclass `BaseAdaptiveTransformer`; this in turn holds the navigation paths the user has taken thus far in an internal DOM[2].

```
<!ELEMENT navigation ((node | lookup-item | adaptive-item)*) >
<!ATTLIST navigation
      selectedNode CDATA #REQUIRED
      domainURI    CDATA #REQUIRED>
 <!ELEMENT node EMPTY>
 <!ATTLIST node
      id           CDATA #REQUIRED
      domainURI    CDATA #REQUIRED>
 <!ELEMENT lookup-item EMPTY>
 <!ATTLIST lookup-item
      lookupId     CDATA #REQUIRED>
 <!ELEMENT adaptive-item EMPTY>
 <!ATTLIST adaptive-item
     adaptationId  CDATA #REQUIRED>
```

Listing 5.3: Navigation generation input structure definition

The transformer uses as input the output of the model filtering described above, which is structured according to the Document Type Definition (DTD) given in Listing 5.3. During transformer processing, the internal DOM is augmented with the nodes provided as input. Any elements named `lookup-item` or `adaptive-item` require special handling. In case of a `lookup-item`, an instance of `OntologyLookupSpec` (see Section 5.2.4) is created for this `lookupId`, which is interpreted as the URI of a lookup specification defined in the application model. An implementation for an *OntologyExplorer* component is then retrieved via the framework and its `explore` method called to find ontology items, that are finally added to the navigation tree. In a similar manner, any element `adaptive-item` is replaced by a list of navigation nodes deemed relevant in this context. For this determination, an *AdaptationEngine* is looked up and its `getRelevantItems` method called with this element's `adaptationId`, which is interpreted as the URI of an adaptation specification in the application model. When all new nodes have been processed, the currently selected path is marked, and the current navigation tree is output as an XML stream to allow for further pipeline processing. The nodes in the tree are identified by ids, which are the URIs of the ontology entries they originated from.

The second phase of navigation processing is shown in the upper half of Figure 5.12 and builds upon the result of the first set of processing steps. The generated navigation tree is read and aggregated with elements from the domain ontology, while filtering for the URIs of the navigation nodes, to read any further information regarding these elements from the ontology, such as labels.

```
<!ELEMENT navigation (node*) >
<!ELEMENT node (info , node*) >
<!ATTLIST node
      id CDATA #REQUIRED
      path CDATA #REQUIRED
      selected (true | false) #IMPLIED>
```

Listing 5.4: Navigation generation output structure definition

The output of this processing is a generic navigation structure in XML form, following the DTD given in Listing 5.4. The child element `info` of a `node` is not further specified in this DTD extract;

[2]Document Object Model

this element contains a copy of the ontology entry of this node in the model and thus contains sub-elements such as the labels, and any further sub-elements used by the modeler. Later processing stages may then, on the one hand, generate content elements appropriate to the current navigation path, and, on the other hand, produce a graphical representation of this navigational structure.

5.3.2 Content Generation

The role of the *ContentTransformer* component is to generate content appropriate to context. The processing steps follow a principle similar to navigation generation (see Section 5.3.1); pipeline processing is detailed in Figure 5.13.

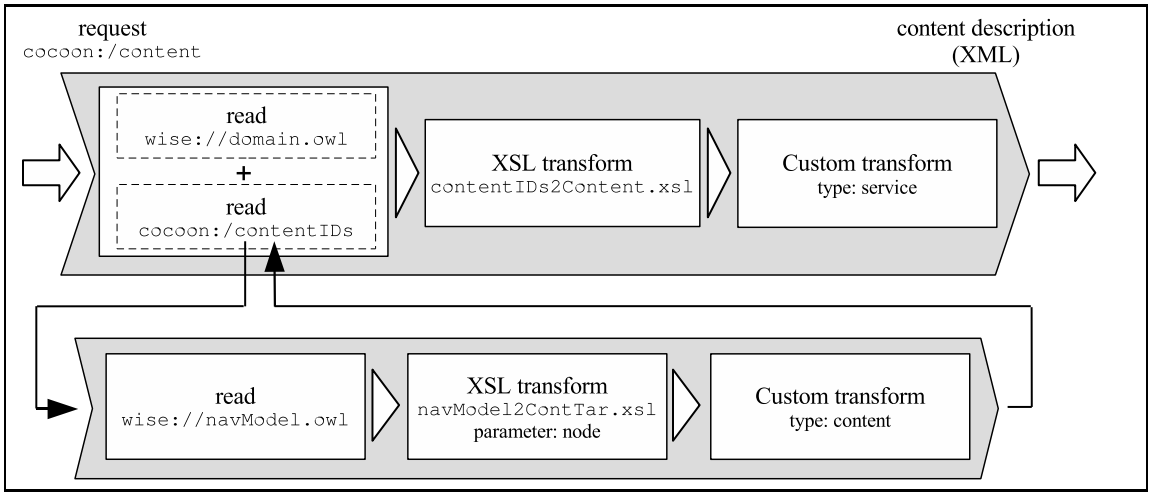

Figure 5.13: Pipeline processing for content generation

An XSLT stylesheet extracts, from the navigation model, content parts that are associated with the node the user has selected in the current request. Some of these parts refer to entries in the domain ontology, via URI. Other parts can be marked as adaptive, meaning they do not explicitly refer to an entry in the domain ontology but should instead be computed according to context; in addition the reference to an adaptation specification is provided. Dynamic content generation is handled by the Java class `ContentTransformer`, registered as transformer class in the Cocoon sitemap under the name `content`. For those content entries that are fixed (i.e., have a reference towards the domain ontology), this transformer passes them through without further processing; whereas for those parts marked as adaptive, the transformer queries the *AdaptationEngine* component for resources according to the adaptation specification and the current context. The *AdaptationEngine* returns several entries identified by URI, referring to an ontology element. As for navigation generation, content generation may also use ontology lookup specifications, to be resolved by an *OntologyExplorer*. The `ContentTransformer` outputs all dynamically gathered parts to the XML stream in the same format as the parts that were not gathered via adaptation.

The second processing stage extracts the domain ontology entries for the computed URIs. The output of the content processing stage follows the DTD given in Listing 5.5. The `ontologyChild` element is used as a placeholder in this definition and is not used as such in the structuring. In place of this element, any child elements modeled in the ontology for this content item are passed through.

The content structures are passed on for processing related to service incorporation, discussed below.

```
<!ELEMENT content (item*)>
<!ATTLIST content node CDATA #REQUIRED>
<!ELEMENT item (ontologyChild*)>
<!ATTLIST item
        id CDATA #REQUIRED
        navContainer CDATA #REQUIRED
        type CDATA #REQUIRED
        source CDATA #IMPLIED>
```

Listing 5.5: Content generation structure definition

5.3.3 Service Incorporation

In the content generation process, any elements referring to services are ultimately processed by a custom transformer registered under the name `service` in the Cocoon sitemap, as indicated by Figure 5.13. This processing is accomplished by the Java class `ServiceTransformer`, which addresses the following aspects of service incorporation:

- if service consumption requires user interaction, the transformer must determine required parameters for this service and generate an XML description for the service such that GUI generation processing has sufficient information;
- if the service must be called in current pipeline processing (for example, consequently to an explicit user request for this service), the transformer calls the Web Service and converts its response into a content description that, again, can be used by GUI generation.

The `ServiceTransformer` listens for any entries representing services; these are entries with the attribute `type` set to `wise:service` (see Section 3.5.1). Via the `AppModel` (see 5.2.3), the service description in the model is read to create an instance of the `ServiceEntry` class. The transformer now reads the parameters of this service and attempts to determine values for these parameters. The HTTP request is evaluated first; this contains a value if the service was called in the previous user interaction. If the request does not already contain a value, a context-relevant value is looked for. As the `ServiceTransformer` subclasses the `BaseAdaptiveTransformer` (see Figure 5.10), it has access to an *AdaptationEngine* component, that is queried in order to determine a value for the parameter. The transformer consumes the incoming content `item` description (see Section 5.3.2) and replaces it with a `serviceItem` as specified by the DTD given in Listing 5.6.

```
<!ELEMENT serviceItem (parameter*)>
<!ATTLIST serviceItem
        id CDATA #REQUIRED
        navContainer CDATA #REQUIRED
        type (wise:service) #REQUIRED
        source CDATA #REQUIRED>

<!ELEMENT parameter (#PCDATA)>
<!ATTLIST parameter
        name CDATA #REQUIRED
        label CDATA #REQUIRED>
```

Listing 5.6: Service generation structure definition

The `ServiceTransformer` then verifies if the service entry currently being processed corresponds to a service that the user has just called in the current interaction step. If so, the Web Service is called using the Apache Axis toolkit (The Apache Software Foundation, n.d.d). The service response is then analyzed regarding its structure and converted into an XML representation encapsulated in a `serviceResponse` element with an attribute named `pattern`. How precisely this output is structured depends on the pattern classification and interpretation used.

To summarize, the `ServiceTransformer` transforms the XML description of a service entry into a representation that includes service parameters and default values; and furthermore, if needed, calls the service and augments the output structure with the results from the service call.

5.3.4 View Filtering

The `ViewTransformer` is responsible for accounting for the requirements expressed in the view model (see Section 3.5.6). The component acts, in essence, as a filter according to context. The application elements input (as an XML representation) to this transformer can have view information associated to them in the view model. When this is the case, the transformer reads the context rule for the view, and asks an *AdaptationEngine* whether this context condition is currently met. If the condition is not met, the transformer discards this application element, as well as any subelements (in the XML sense). Other elements are not affected by this transformer.

5.3.5 Presentation Information Generation

The `PresentationTransformer` is a Java class implementing the final custom transformation component in the CATWALK processing sequence (see Figure 5.1), and is responsible for determining how the application's response to a user request should be presented. This determination, as the other stages of application generation, can make use of context information.

The `PresentationTransformer` assesses for each item to be rendered the navigation container it originated from, and looks up the presentation model (as defined in Section 3.5.6) for any presentation information on this container.

If the presentation style is to be determined by adaptation, the `PresentationTransformer` looks up an *AdaptationEngine* component and asks it to provide adaptive items corresponding to the adaptation specification; whereas the usual case in such an adaptation scenario is expected to be that only one item is returned and that it refers to a style entry in the presentation model. If no adaptation is specified, or adaptation did not find an appropriate style entry, any fixed style association present in the model is used instead. As a result, the transformer augments the XML representation of the content elements (see Section 5.3.2) by adding an attribute `presentationId` that refers to a style entry of the presentation model.

The XML representation of the content as output by this component is usable for GUI generation. If the style entry refers to a template, a corresponding XSLT stylesheet is applied. If the entry refers to a CSS class, a reference to this class can be generated in the final XSL transformation.

5.3.6 Summary

Figure 5.14 summarizes processing for generation of a response to a typical user request. The diagram illustrates flow in the main processing steps; for clarity, additional required mechanisms such as model and context state accesses are not shown.

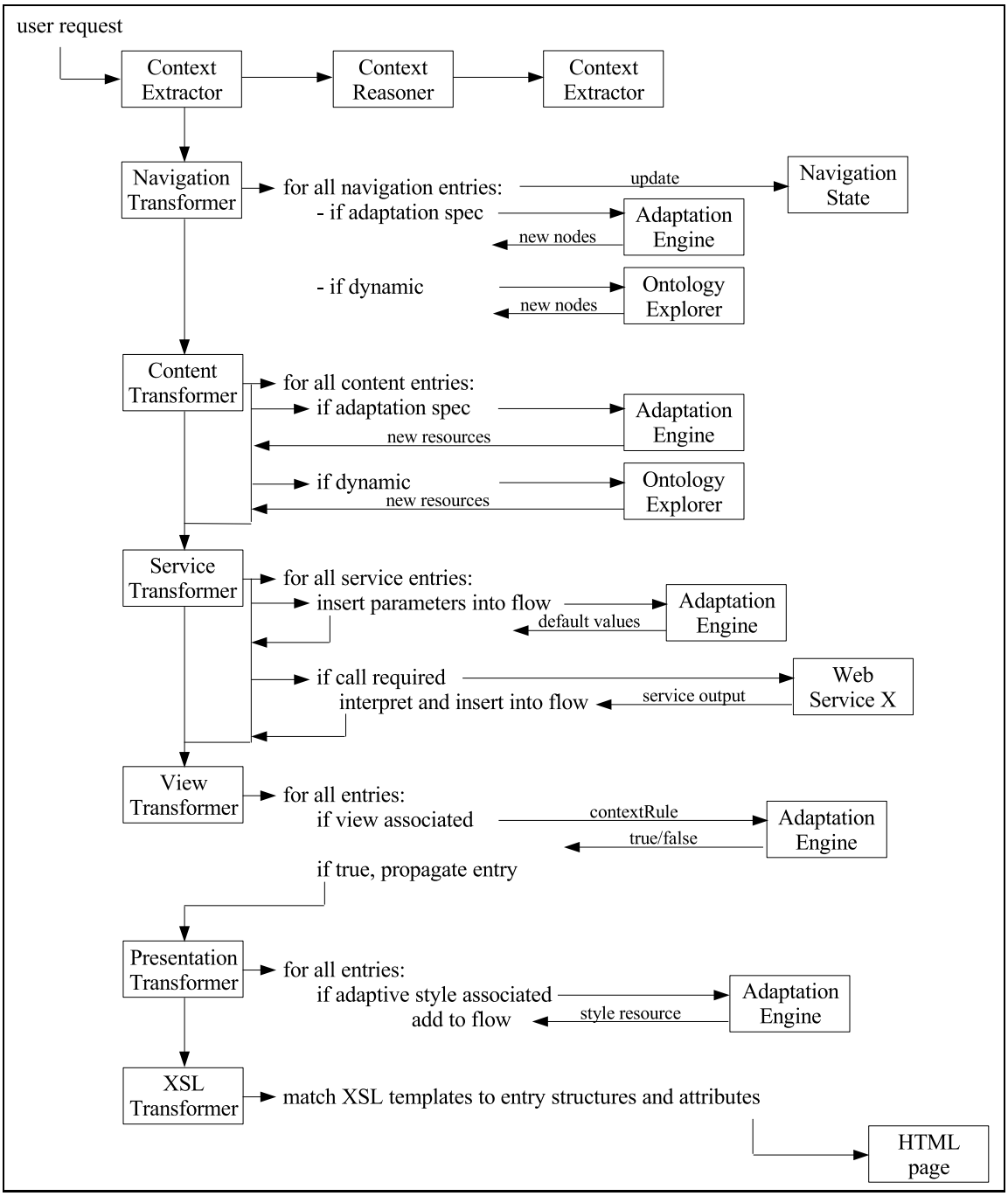

Figure 5.14: Processing flow for response generation to a typical user request

5.4 Context Simulation

CATWALK allows for simulation of context; meaning a user (for example, in the role of an application tester) can explicitly say what the context should be, thus overriding sensed context factors. This feature allows for testing of the adaptation effects of the Web application; and, in a larger sense, can be used as a tool to attempt to evaluate effects of context-awareness in Web applications.

The set of context simulation commands issued by the user is stored in the current session. After regular context sensing mechanisms (see Section 5.2.2) are applied, the context state is overwritten with the simulation commands. An example for such a set of commands is:

- deactivate context factor "intranet" in category location;
- activate context factor "internet" in category location.

The user navigating in a CATWALK-based adaptive application can choose to activate a context simulation GUI. This window shows context state and context history. Furthermore, context factors present in the context model are shown, and the user can select individual factors to decide whether to activate or deactivate them, thus allowing for simulation of context.

An example GUI for context simulation based on a case study is shown in Section 5.5.3. Implementation details are provided in Appendix D in the form of a class diagram (Figure D.3) and pipeline processing (Figure D.4).

5.5 Case Study: Example Web Application

The case study describes a prototypical Web application whose purpose is to illustrate what adaptation effects context-awareness can trigger in a Web application, and how these can be modeled as defined in Chapter 3. The CATWALK framework interprets the models and generates the Web application, using the mechanisms previously described in this chapter. The models used in this case study are provided in OWL form in Appendix C.

We begin by illustrating how the core features of navigation and content generation are achieved, independently of context considerations, before proceeding to discuss adaptation effects.

5.5.1 Application Generation

Figure 5.15 illustrates the core features of navigation and content generation. In the screenshot, the user has followed a certain path through the navigation model; the resulting tree is shown on the left side, the chosen nodes are highlighted. Each node corresponds to an entry in the domain ontology. Content associated to the currently selected node ("Sports shoes") is presented on the right side.

This screenshot illustrates several possibilities for application modeling, the model itself is given in Appendix C. For the "Home" node, the nodes "Wear", "Stores", etc. are modeled as fixed navigation targets. For the "Shoes" node, an ontology lookup is specified to provide sub-nodes for navigation; this lookup specifies to lookup subclasses of "Shoe" up to a depth of 1, therefore resulting in the three sub-nodes shown. When navigating to the "Sports shoes" node, the navigation model states that one fixed piece of content is associated (a document, presented as a link to access the document), and one ontology lookup is used to retrieve further content: in this case, the lookup is specified to retrieve any domain ontology items in the relation `related-content` with the ontology item the current node is based upon, resulting in the selection of a piece of text. A further possibility for page composition, not shown in this screenshot, is to provide a free-hand query: in the model, this is illustrated for "Special offers", to which a query is defined to find items related to this concept.

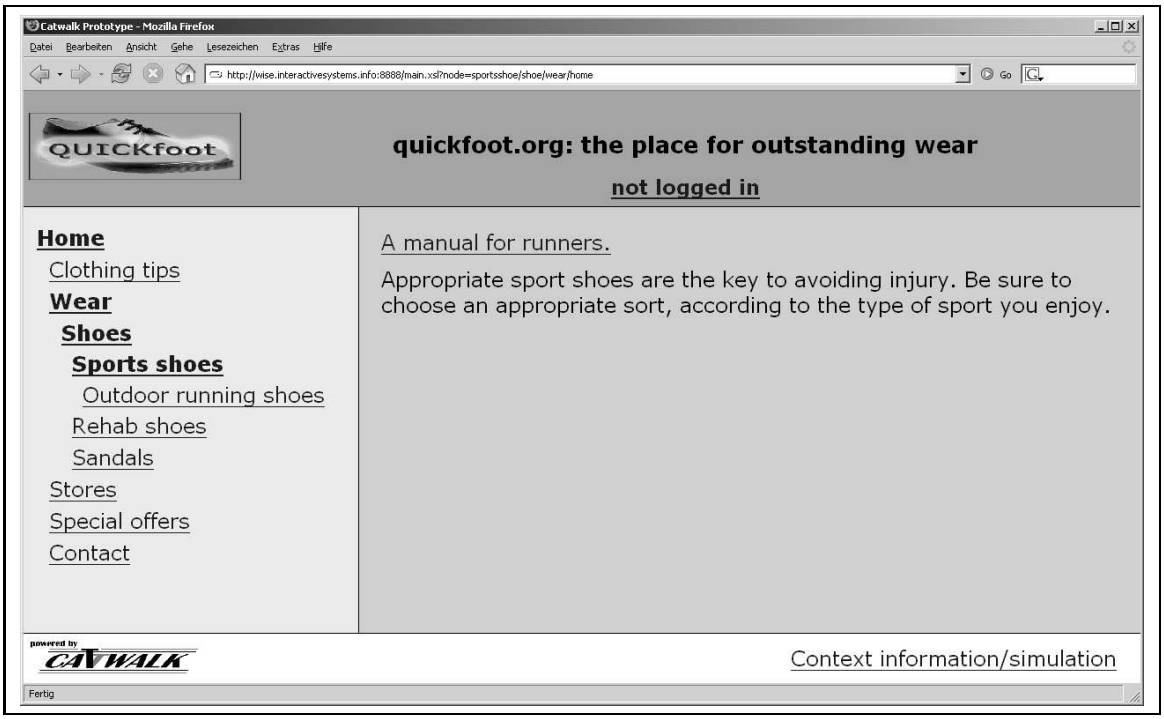

Figure 5.15: Case study: application screenshot

5.5.2 Adaptation Effects

A graphical representation of context factors used in the prototype and their relation to application model elements is given in Figure 5.16; the graphical representation is to be interpreted as discussed in Section 3.3.4. This figure is an extract of the essential items involved in the demonstration of adaptation.

Navigation Adaptation

An example for navigation adaptation is given in Figure 5.17. Presume the context “summer”; i.e., in the *Time* context ontology the instance “summer” of the class “Season” was activated.

The navigation model contains, for the navigation node labeled as “Clothing tips”, an adaptation specification defined to retrieve domain elements that can be offered as links. As a consequence, CATWALK adds within this navigation node any navigation possibilities that are considered context-relevant, in the order of relevance. The node labeled as “Outdoor running shoes” is considered most relevant, as it is in an immediate context relation with “summer” in the model. The adaptation block properties in this scenario instruct the system to explore super-class relations up to a depth of two, therefore two levels of super-classes of “Outdoor running shoes” are also retrieved: in this case, these are the nodes labeled as “Sports shoes” and “Shoes”. Each level induces a loss of relevance (whose percentage is also given in the adaptation specification), and thus the further away in the hierarchy, the less relevant elements become.

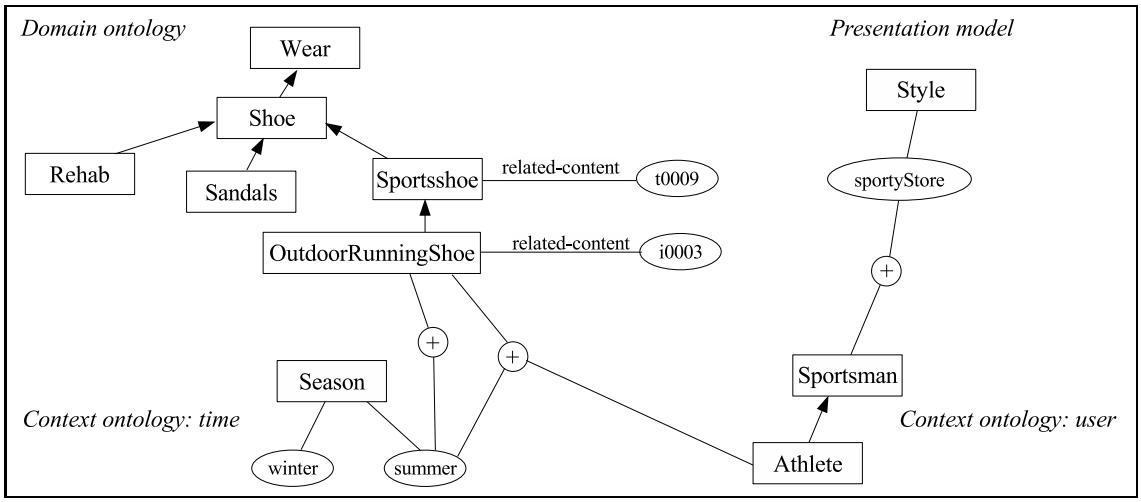

Figure 5.16: Case study: model extract

Figure 5.17: Navigation adaptation demonstration

Content Adaptation

Composition of content for the node "Wear" is defined in the navigation model, and associates a fixed text from the domain ontology, and a document, which is made available via a link. Furthermore, an adaptation placeholder is associated to this node. The resulting page is shown in Figure 5.18. The adaptation specification of this placeholder defines to retrieve context-relevant domain elements, and any elements associated to these via the `related-content` property. Furthermore, if any elements are found, a fixed content piece from the domain ontology is specified to be added prior to the elements. In the example, the concept "Outdoor running shoes" is context-relevant; this is presented as a link in the content area, allowing the user to directly navigate to this navigation node (thus avoiding multiple steps of navigation in a menu). Furthermore, an associated image is context-relevant and presented.

Figure 5.18: Content adaptation demonstration

Service Adaptation

For the illustration of service adaptation, a Web Service representing the functionality of looking for auctions across several auction houses is used. This Web Service provides a "search" operation that can be parameterized with 3 values: category, season and size. The Web Service is implemented with Apache Axis and made available on the network. In the domain model, the service's existence is specified, as are contextual hints for the service's parameters. For this service, the parameter "category" is associated with an entry in the domain ontology (the concept "Wear"); the "season" with the "Season" class of the *Time* ontology (as a category of context); "size" is associated with an attribute that is not defined more specifically.

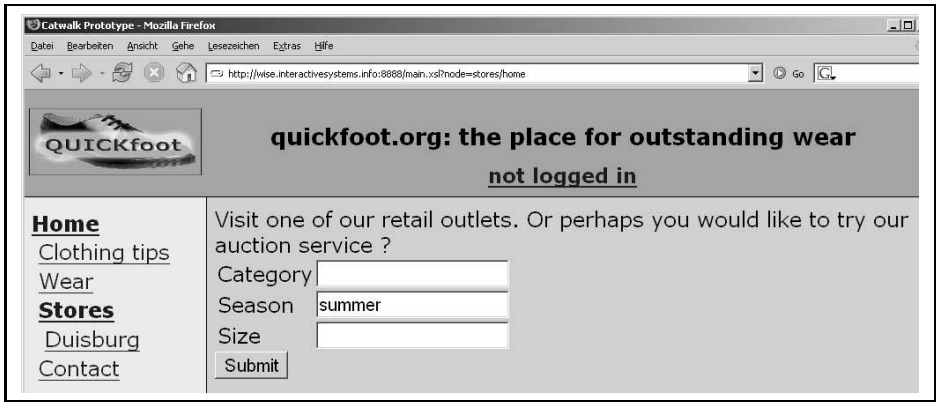

Figure 5.19: Service adaptation demonstration 1

The Web Service's functionality is presented when the user navigates to "Stores", as shown in Figure 5.19, because the service is associated to this node in the navigation model. The *AdaptationEngine* tries to determine appropriate default values for the service parameters. For the parameter with label "Season", "summer", as a currently active instance of the appropriate class from the *Time* ontology, is used as value. Regarding the other parameters, the *AdaptationEngine* did not find appropriate values. The interface for calling the service is rendered with an XHTML form consisting of text

areas, in which the context-relevant parameter value is entered as default. A more elaborate user interface can propose, for example, drop-down lists to choose from possible values, wherein the default value is activated according to context. The user interface should also display (language-sensitive) labels for the default parameter values, for those cases where the values refer to an ontology entry; for simplification the ontology ids are used here instead.

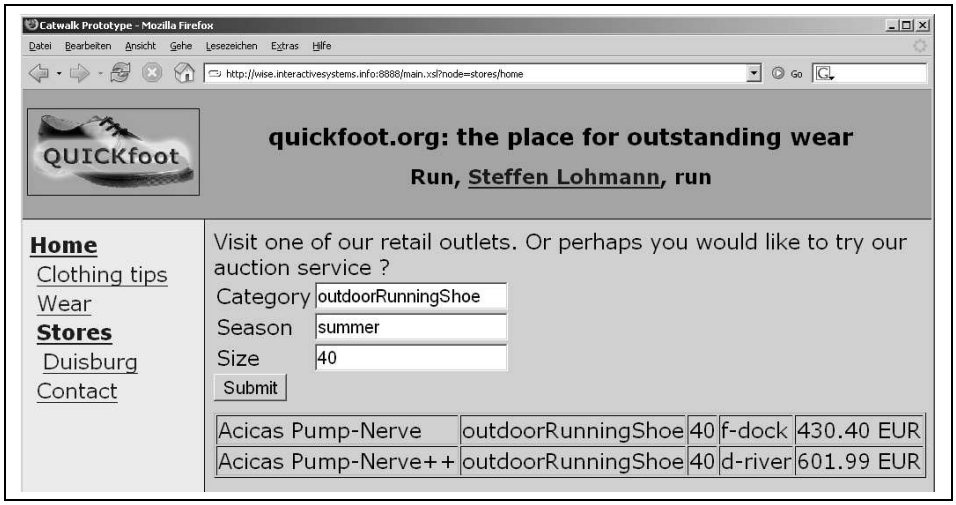

Figure 5.20: Service adaptation demonstration 2

Consider now a further-going scenario, in which the user has recently viewed product information on "Outdoor running shoes", either by menu navigation or by following a recommended link as seen in Section 5.5.2. Furthermore, the user has logged in to the system. The *AdaptationEngine* can now provide more default parameters, as shown in Figure 5.20. The "Category" field contains as default value the most recently visited category of wear; in this case the "Outdoor running shoes", being a sub-category of wear. To determine a value for the "Size" field, the *AdaptationEngine* searches for an active value for a "size" attribute in any of the context categories; in this scenario a value is found within the attributes of the currently logged-in user. The user may override any of the default parameters. In Figure 5.20, the user has furthermore called the service (without changing the default values); the result is rendered according to the "table" user interface template.

These examples illustrate further issues regarding user-friendly binding of services and domain knowledge (see also Section 4.3.3). Generally speaking, service parameter values can be automatically extracted from context, as demonstrated above. For an optimal binding however, more intelligence is required. For instance, the "Size" parameter is known from the attributes of the logged-in user; however if several attributes representing size (for example shoe size and shirt size) are present, more knowledge is needed: does this service expect size values regarding shoes specifically, or, more likely in this case, should the value used depend on the product category which size should be retrieved? If so, these types of sizes would require an ontological connection to the product category hierarchy so that such a determination can be made. In fact, this sort of "richly-typed" connection between domain knowledge and Web Services is not intrinsically related to the issue of context-awareness, but is, like context-awareness, a prerequisite for an optimal service binding in the sense of user-friendliness.

Presentation Adaptation

Adaptation regarding presentation, in the largest sense, can determine which elements should be viewable in the current context, and furthermore how the elements are presented (that is, the selection of GUI attributes).

Figure 5.21: Presentation adaptation demonstration - View

Which elements are viewable in which context can be guided by the view model (see Section 3.5.6). In the example illustrated in Figure 5.21, a view is defined on a content piece associated to "Wear". This content element refers to a file containing a (potentially large) catalog of products in PDF format. The view defines that this content element should only be offered when the user is using a desktop browser, as the element is not deemed useful for viewing on a device with more limited memory and display resources, such as a PDA. On the left-side of the figure, the application is shown when the hardware device context factor "desktop browser" is active; the content element mentioned above is accessible via a link. On the right side, the context factor "pda" is set instead (via context simulation); accordingly, the content element is not accessible.

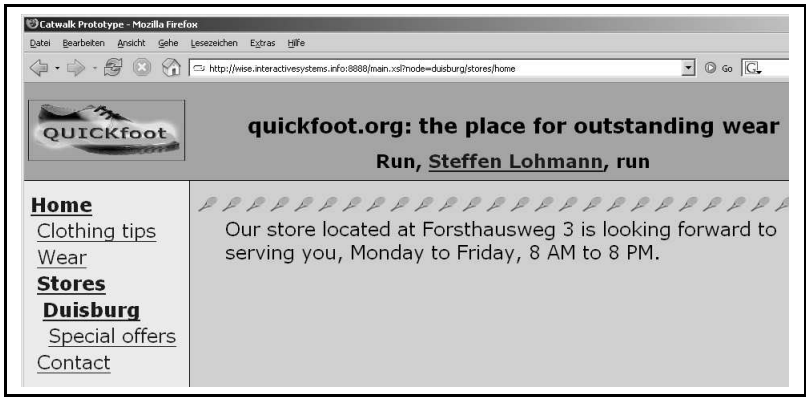

Figure 5.22: Presentation adaptation demonstration - GUI attributes

The presentation model, in turn, can associate GUI attributes with application elements. In the example illustrated in Figure 5.22, the underlying presentation model associates a presentation definition with the navigation container reached via the navigation node labeled as "Duisburg". This presentation definition has an adaptation specification that states that a presentation model element should be determined according to context.

The application model expresses that a sportive appearance is appropriate when a user known as a sportsman navigates to the Duisburg store. Otherwise, a more neutral style is to be used. The scenario shown in Figure 5.22 presumes a context setting where the user is identified as a sportsman; i.e., context factor "Sportsman" of the user context ontology is active. For a "Sportsman", the context model specifies the "sportyStore" style element from the presentation model as being relevant. The adaptation mechanism thus retrieves this style element, that in turn refers to a CSS class. The content element is therefore rendered with attributes defined in that CSS class.

5.5.3 Context Simulation

A user, for example acting in the role of an application tester, can choose to activate the context simulation GUI when navigating in the generated application. Context simulation appears as an independent pop-up as shown in Figure 5.23. The GUI shows context state and context history. Context factors available from the context model are presented in drop-down lists, allowing the user to select individual context factors to explicitly activate or deactivate them.

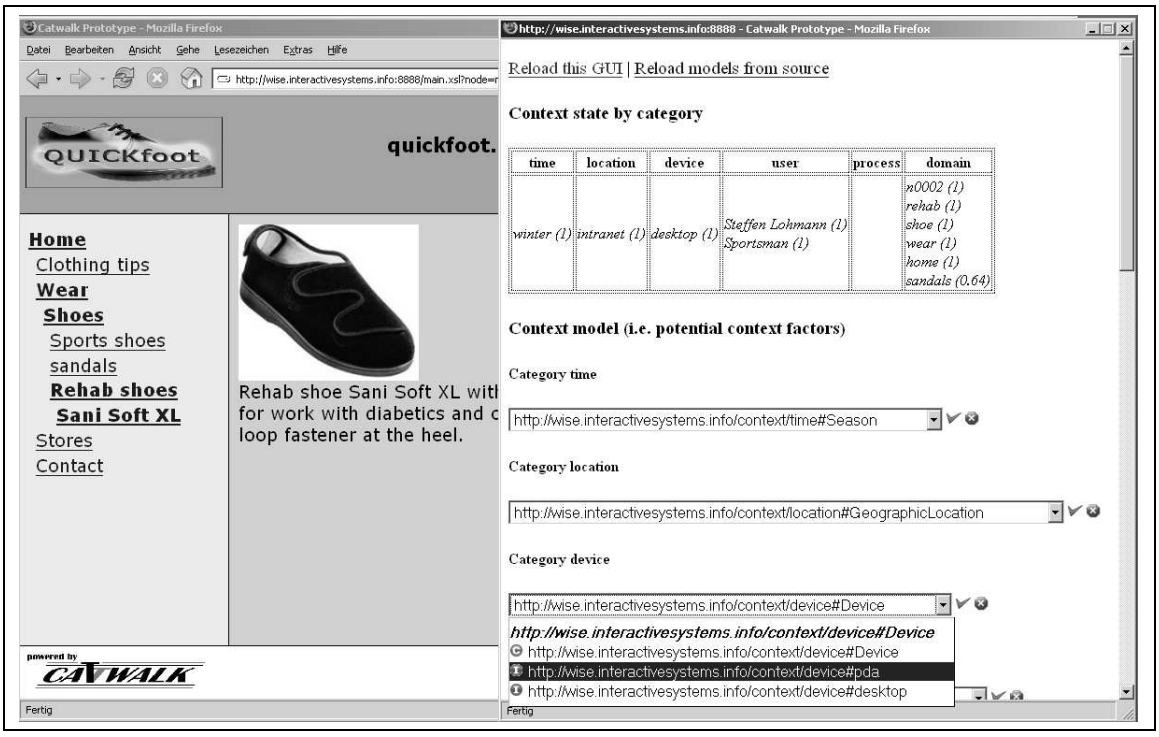

Figure 5.23: Context simulation screenshot

5.6 Discussion

The main purpose of the CATWALK framework is to facilitate and systemize use of adaptation within Web applications. Corresponding applications can offer functionality in an adaptive manner, the goal of which is to enhance interaction of users with the software's offerings.

Functional Aspects

Development of context-aware Web software is supported by application models integrating knowledge about context, and facilitated by a system architecture augmented by a context framework that generates a dynamic, adaptive Web application based on the models. The Web application development process can thus extend conventional features with adaptation functionality by using the methods described in Chapter 3, and by basing the run-time application on the software framework described in this chapter. This development process can involve several roles:

1. context-modelers define context factors within context ontologies and their relations with the application model;
2. adaptation modelers specify where adaptation is to occur in the application, what sort of adaptation is desired and what strategies are to be used;
3. context-sensing implementors configure, complete or extend the components responsible for sensing;
4. adaptation implementors customize the adaptation component and its usage of ontology inference mechanisms;
5. application testers and quality experts use the generated application as a prototype along with the context simulation functionality to verify the effects of context on the application.

Roles 3 and 4 are optional: the framework contains a default implementation and interpretation for sensing and adaptation. This means that adaptive Web applications can be achieved by means of context and navigation description only, without requiring additional programming. This in turn allows for rapid prototyping of adaptive Web applications; the prototyping is further supported by the possibility of context simulation. Web engineers are provided with a platform supporting observation of the effects of context modeling and adaptation strategies on the actual Web applications; in other words, the recommendation that a design environment should support an "experimental feedback loop, which facilitates the evaluation activity and process recursion" (Nanard and Nanard, 1995) is followed. Rapid prototyping allows iterative user-oriented evaluation, while model-based design facilitates "design-oriented evaluation" (Garzotto et al., 1995).

When a different context-recognition or adaptation mechanism is required, the necessary custom programming is reduced to the essentials of that functionality. An architecture and a default implementation for components for context are provided; these are furthermore based on an existing component framework (Excalibur), therefore allowing to specify in the application configuration which actual software implementations accomplish the given component roles in the run-time environment.

Design Aspects

The work presented in this dissertation is designed to support context-awareness in Web Engineering, that is, in a systematic approach to Web application development. For this purpose, knowledge about

context is structured and, along with domain knowledge and the relation of this domain knowledge to context, represented in (formal) ontologies, as are the individual application models. The model as a whole is uniform, thus permitting uniform model handling and usage of common mechanisms for the various stages of application generation.

The system architecture is designed to make use of modeled knowledge; each functional concern in this architecture is identified and handled separately. Specifically, to support a development process where responsibilities are clearly defined and to provide a system where implementations are extensible, the approach presented in this dissertation consistently follows the "separation of concerns" (SoC) design principle within the engineering process, at various levels:

- design and modeling: the domain ontology as such is modeled, and context factors relevant to the domain are defined within context ontologies. The Web application model is established, based upon the domain ontology. Context relations between elements of the application model are defined separately. Adaptation specifications define a strategy for an adaptation; these specifications can in turn be referenced at those places in an application model where adaptation is desired;

- architecture and implementation: the underlying architecture (Excalibur and Cocoon) follows the SoC principle; the framework for context extends this architecture to account for context-awareness and adaptation, while remaining pluggable to allow for customization of the default mechanisms for context recognition and adaptation.

The approach is extensible, consequently to the choice of software techniques and the conceptual design. On the one hand, as discussed above, the component-based approach to adaptation allows to replace or extend the way each concern is implemented as a software component. On the other hand, all application knowledge is specified in ontologies, therefore with formal semantics. This is a prerequisite for reasoning support; that is, for allowing for complex inferences to be mechanically achieved. Existing reasoners such as FaCT (Horrocks, 1998) or RACER (Haarslev et al., 2004) can be plugged in via JENA to further support adaptation. In addition, the chosen content-based architecture (based on XML) fulfills requirements of Web applications, by addressing the information-centered nature of a Web application, while at the same time enabling operational features by way of Web Service integration. A service description is treated as a piece of content; therefore content-based methodologies can be used to describe parts of the system, whereas context information can be modeled in addition as described in this work.

Technical Aspects

In CATWALK, the two separate concerns of dynamic Web application generation per se and adaptation are handled using two separate mechanisms. The generation is handled as a sequence of XML transformations. XSLT stylesheets are used to filter incoming XML, such as an ontology specified in OWL, and to aggregate results from a previous step with parts of the application model, for example to determine which content containers are to be used at this stage. Each generation phase is, in addition, supported by a custom XML transformer, implemented in Java, which allows for the integration of adaptation results where specified. Adaptation functionality, as is context recognition functionality, is implemented as a separate component, which can be accessed from the transformer components.

The pipeline processing, as a sequence of XML transformations, thus uses a combination of Java and XSL technology. XSL, compared to Java, is more concise when manipulating XML, and processing statements specified as XSL are easier to change, being interpreted at run-time and therefore

not requiring application restart. A significant disadvantage of XSL in this approach is that exact knowledge of the syntax of a language is necessary if XSLT is used for processing it. This is problematic in particular for ontologies, as an ontology can be represented in differing syntaxes. Accessing ontologies via an API (such as JENA) abstracts ontology syntax, and furthermore provides built-in knowledge about the structures, such as transitivity of superclass relationships.

A potential danger in the consistent separation of code according to concerns is that integration and sequencing of components appears complex; see, for example, Figure 5.13. However, these mechanisms are realized with existing features in a public framework (Apache Cocoon); information regarding the underlying mechanisms is thus publicly documented, and the techniques are reusable. An alternative software architecture that would not follow the SoC principle, yet fulfill the same requirements, would ultimately require the same complexity, whereas in that case a higher complexity would be transposed into the software components themselves.

Whatever the technical choices, connections between components and processing steps should be clearly documented; as illustrated in this chapter, UML diagrams (such as class diagrams and sequence diagrams) are helpful in this regard. To document pipeline designs, no standard notation exists and thus this dissertation proposes a notation as shown in the above-mentioned Figure 5.13.

Weaknesses

Support for matching arbitrary environment information onto the context ontologies of a specific application model is limited in the approach presented in this dissertation. Supporting arbitrary sensing in a general-purpose, unifying approach is problematic. In particular, there are no existing Web standards specifying how environment information is defined and how a Web client would communicate this information to the Web server. Current approaches must thus either define a specific protocol and language for transmitting client context information in an application-relevant way, or, as in the approach followed in this dissertation, require custom programming for context matching when the required information does not conform to generally valid patterns (for example, seasons in the category *Time* or browser type in the category *Hardware Device*). However, such a task is facilitated when the provided framework classes are extended and when configuration mechanisms are used as explained in Section 5.2.2. Likewise, custom programming is required to support integration of data originating from any additional, system-external environment sensors.

Chapter 6

Summary and Future Directions

In this chapter, we summarize the engineering method for adaptive, context-aware Web applications presented in this dissertation, and discuss future opportunities for research.

6.1 Summary

The growing complexity of Web-based systems, resulting notably from an increasing information and service offering, poses problems to users and application providers. On the one hand, the increased offering can confuse users regarding what is relevant for them, which makes the application more difficult to use and impedes joy-of-use. On the other hand, this situation also means the application provider's interests are not served. The provider would like to guide the user towards a certain resource selection appropriate to the user: in consumer-oriented applications, to increase the likelihood of consumption and customer loyalty; in a professional environment, to increase user acceptance and productivity.

An approach to address this situation is the adaptation of the application, at run-time, to the context of usage. To ultimately achieve such support, the following challenges and requirements must be met:

Requirements concerning the conceptual approach: The challenges already existing for Web Engineering are compounded by the requirement to account for context and to define adaptation according to this context. This requires a precise definition of context, and an understanding of its key characteristics. Information about the context of usage of an application must be represented in a formal model that is usable in a Web Engineering process. The corresponding analysis and design processes themselves must be structured and systemized.

Requirements concerning system functionality and design: A system supporting an adaptive Web application must dynamically generate a response for a user request to the application. The generation process must be able to function independently of context considerations, whereas each phase of the generation process can in addition be influenced by the context. Therefore, the system must be context-aware; it must interpret environment information as context knowledge, and use the conceptual information about context to support adaptation at the various levels of the Web application: navigation, resource selection and integration, and presentation. This Web application may offer the user, in addition to information selection and presentation, operational features.

State of the art: The question of what sort of adaptation is effective, to what degree, in which situations and with which kinds of mechanisms, is currently difficult to address in a systematic manner. A review of existing work shows that there is no established Web Engineering process which accounts for context comprehensively, while producing artifacts usable by a run-time system. Notably, existing approaches focus either on the design phase of engineering, or focus on a particular aspect of a run-time system, such as context sensing. In the first case, resulting applications are disconnected from the original models, making rapid prototyping and the comparative studies of varying context representations and mechanisms difficult. In the second case, the lack of a systematic (engineering) approach means that resulting applications are ad-hoc, particular solutions, difficult to maintain and reuse.

Contributions: Building on this problem analysis, the major contributions of this dissertation can be summarized as follows:

- A conceptual framework for context represents the underlying information regarding context of usage of an application (Chapter 3). A context model in this framework is understood as a combination of semantic and associative networks, using ontologies for knowledge representation. This model is integrated in a general-purpose, compositional approach to Web application development (Section 3.4), in which adaptation effects can be specified according to the context model and ontological knowledge (Section 3.5).

- A specific Web software architecture for context is proposed (Chapter 4). Systems conforming to this architecture are capable of interpreting application and context models in order to support adaptive applications. The architecture supports operational features via the incorporation of Web Services, whose selection and integration within the user interface is guided by context.

- A software framework supports the engineering of adaptive Web applications according to the method presented in this dissertation. The presented framework, called CATWALK (Chapter 5), is an implementation of the architecture for context described in Chapter 4. The framework consists of a set of components, each component being responsible for a specific concern, and of a sequencing of these components which generates an adaptive Web application. The framework contains default implementations for context inference and adaptation mechanisms, and thus supports model-based prototyping of adaptive Web applications. The usage of the framework is illustrated in a case-study describing an e-commerce scenario. Indeed, the method presented in this dissertation can be used in particular regarding systemization of the engineering of adaptive e-commerce systems, to formalize usage of user profiles and associated adaptations, and to enhance adaptation by taking into account further context factors.

- The engineering method for adaptive, context-aware Web applications presented in this dissertation follows the principle of separation of concerns, in order to support a Web Engineering process accounting for context in which people with different expertise participate (Section 5.6). Domain, application, context, and adaptation concerns can be modeled in separate, succeeding steps. The system supporting the adaptive application at run-time is designed such that system concerns are handled separately as components; these components either support context-awareness or adaptation, or achieve a step of the generation process. This design allows for customization, meaning that application developers specializing in a particular aspect of context-awareness, adaptation, or application generation, can extend or replace the manner in which each concern is implemented, without altering the other aspects of the architecture or

requiring code changes in the application itself. Application testers can evaluate the effects of individual context factors in the generated prototype.

- The generic nature of the conceptual framework for context and the component-based system architecture support the implementation and comparative evaluation of various context-inference and adaptation algorithms and strategies.

Table 6.1 summarizes how information about context is represented and used in the approach presented in this dissertation.

	explicit	*implicit*
context knowledge	• semantic network (context ontologies) • associative network (context relations)	• relation following during adaptation (via adaptation spec) • relation following during context inference
context usage	• model specification according to context rules • value specification according to context hints	• computation of context-relevant ontology items

Table 6.1: Accounting for, and using context: summary

6.2 Future Directions

This dissertation introduces models for Web applications, a corresponding architecture, and a software framework enabling rapid prototyping of context-aware, component-based Web applications. Future research can use and extend this approach along several directions.

Supporting Application Modeling

For large-scale scenarios, a user modeling a Web application and relevant context factors needs to be supported by appropriate software.

System support for the modeling process includes tools for visualization and editing of complex networks. In addition, modeling can be facilitated, on the one hand, by usage or creation of context catalogs pertinent to specific business domains, and, on the other hand, by the system generating proposals for influence relations based on data mining. In this regard, the approach of this dissertation provides a basis by defining context factors within ontologies, and supporting their association with a domain ontology, thus allowing the use of common vocabularies and mechanisms.

Enhancing the Context System

The architecture for context presented in this work, and in particular the CATWALK system, provides a framework for achieving adaptivity in Web applications. Default implementations for core components are discussed in this work. Future research can enhance this system to provide further functionality or to plug-in alternative implementations and interpretations for context.

Further functionality could notably be integration with distributed agents that would be responsible for sensing the physical environment, thus enabling richer context sensing than that available by

using standard HTTP data alone. Alternatively, extensions to HTTP or standardization attempts for richer Web client to Web server communication can be studied, such as (W3C Device Independence Working Group, n.d.) that can be used for transmitting more elaborate client information.

Some context information is irregular and difficult to predict, and therefore difficult to model explicitly. For such context information, it may be helpful to have a system that "learns" over the course of operation, to make more accurate assessments of context over time. One possibility is to use Hidden Markov Models, as described, for example, in (Yonezawa et al., 2001). In addition, context influence relations may be refined or created during system operation. Possible approaches are to track user behavior regarding the adaptively proposed links; or to gather user feedback regarding context relations during system operation, in analogy to the proposal described in (Staff, 2002) that recommends gathering user feedback regarding which documents are relevant to each other.

The described framework is a basis for studying alternative context mechanisms, as discussed in Section 5.6. Notably, context-inference plug-ins can be added declaratively to the run-time system in order to study implications of other techniques for inference and to research their effectivity within adaptive Web applications.

As stated, the focus of the CATWALK system is to support rapid prototyping of adaptive Web applications. For the goal of supporting rapid application development for potentially commercial-grade software, a comprehensive definition of user interface patterns and their application to specific information structures is required, where patterns in this sense are understood as specifications that are interpretable by the system. In addition, a seamless integration of interactive services requires to address the issue of semantic interoperability in more depth. In this regard, two general directions of research are conceivable: development of custom catalogs to explicitly realize "contextual interoperability" (Strang et al., 2003b), or integration of standardized catalogs, upon which context ontologies, matching mechanisms and services would be built.

Usability

The goal of adaptivity as outlined in this work is to present the end-user relevant information and application navigation possibilities, and to provide relevant interactive services. The assumption is that providing Web applications tailored to user needs raises productivity and joy-of-use of people working with these applications.

This dissertation presents a method for achieving dynamic Web applications that may be adaptive. The engineer defines which application elements shall adapt according to context. In addition, context can be simulated (or deactivated) in the run-time system. This work thus provides a basis for conducting general-purpose usability studies regarding use of context, notably for comparing effects of using classic (non-adaptive) mechanisms with the use of adaptation mechanisms. Future work can therefore attempt to quantify usefulness of adaptation, in particular, to ascertain for what purposes and in which application situations an adaptation mechanism in fact enhances user productivity or enjoyment. This, in turn, is a prerequisite to address the issue of cost-benefit in engineering, meaning, to determine in which situations the added effort required for context and adaptation engineering yields a significant benefit.

Appendix A

Standards and Standard Authorities

This appendix provides an overview of standards mentioned in this work; whereas standard is understood as something established by an authority. In Web Engineering, several such authorities exist; those authorities particularly relevant to this work are summarized in Table A.1. The standards are listed in Table A.2.

Table A.1: Standard authorities

Acronym	*Name*	*Homepage*
DCMI	Dublin Core Metadata Initiative	`http://dublincore.org`
IETF	The Internet Engineering Task Force	`http://www.ietf.org`
ISO	International Organization for Standardization	`http://www.iso.org`
JCP	Java Community Process	`http://jcp.org`
OASIS	Organization for the Advancement of Structured Information Standards	`http://www.oasis-open.org`
OMA	Open Mobile Alliance	`http://www.openmobilealliance.org`
OMG	Object Management Group	`http://www.omg.org`
W3C	World Wide Web Consortium	`http://www.w3.org`

Table A.2: Standards

Acronym	*Name & Reference*	*Authority*
CSS	Cascading Style Sheets `http://www.w3.org/Style/CSS/`	W3C
DocBook	The DocBook Document Type `http://www.oasis-open.org/docbook/specs`	OASIS
Dublin Core	DCMI Metadata Terms `http://dublincore.org/documents/dcmi-terms/`	DCMI
HTTP	Hypertext Transfer Protocol `http://www.ietf.org/rfc/rfc2616.txt`	IETF

Table A.2: (continued)

ISO 9126	Information technology - Software Product evaluation - Quality Characteristics and Guidelines for their Use Standard ISO/IEC 9126	ISO
JSR-53	JavaTM Servlet 2.3 and JavaServer PagesTM 1.2 Specifications http://www.jcp.org/en/jsr/detail?id=053	JCP
JSR-54	JDBC 3.0 Specification http://www.jcp.org/en/jsr/detail?id=054	JCP
JSR-168	Portlet Specification http://www.jcp.org/en/jsr/detail?id=168	JCP
JSR-901	Java Language Specification http://www.jcp.org/en/jsr/detail?id=901	JCP
OWL	Web Ontology Language http://www.w3.org/TR/2004/REC-owl-ref-20040210/	W3C
RDF	Resource Description Framework http://www.w3.org/RDF/#specs	W3C
SGML	Standard Generalized Markup Language ISO 8879:1986	ISO
SMIL	Synchronized Multimedia Integration Language http://www.w3.org/TR/REC-smil/	W3C
SOAP	Simple Object Access Protocol http://www.w3.org/TR/soap12-part1/	W3C
UDDI	Universal Description Discovery and Integration http://uddi.org/pubs/uddi_v3.htm	OASIS
URI	Uniform Resource Identifiers http://www.ietf.org/rfc/rfc2396.txt	IETF
URL	Uniform Resource Locators http://www.ietf.org/rfc/rfc1738.txt	IETF
WSDL	Web Services Description Language http://www.w3.org/TR/2001/NOTE-wsdl-20010315	W3C
WSRP	Web Services for Remote Portlets Specification http://www.oasis-open.org/committees/wsrp	OASIS
XML	eXtensible Markup Language http://www.w3.org/TR/xml11/	W3C
XML Schema	XML Schema http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/	W3C
XPath	XML Path Language (XPath) 2.0 http://www.w3.org/TR/xpath20/	W3C
XSL	The Extensible Stylesheet Language Family (XSL) http://www.w3.org/Style/XSL/	W3C
XSLT	XSL Transformations (XSLT) Version 2.0 http://www.w3.org/TR/xslt20/	W3C

Appendix B

Model Elements Type Definition in OWL

This appendix provides the definitions, in OWL, of the types of elements which can be used in a Web application model as described in this work. See Section 3.5 for further explanations.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
    <!ENTITY wise_base  "http://wise.interactivesystems.info/wise" >
    ]>
<rdf:RDF
  xmlns:owl  ="&owl;"
  xmlns:rdf  ="&rdf;"
  xmlns:rdfs ="&rdfs;"
  xmlns:xsd  = "&xsd;"
  xmlns      = "&wise_base;#"
  xmlns:wise = "&wise_base;#"
  xml:base   = "&wise_base;"
  >
  <owl:Ontology rdf:about="">
    <rdfs:comment>Core types for application modeling</rdfs:comment>
    <owl:versionInfo>$Id: core-types.owl 621 2006-03-25 17:48:18Z kaltz $</owl:versionInfo>
  </owl:Ontology>

  <!-- Content-piece type definitions -->

  <owl:Class rdf:ID="content"/>

  <owl:DatatypeProperty rdf:ID="source">
    <rdfs:domain rdf:resource="#content" />
    <rdfs:range  rdf:resource="&xsd;anyURI"/>
  </owl:DatatypeProperty>

  <owl:Class rdf:ID="text">
    <rdfs:subClassOf rdf:resource="#content" />
    <rdfs:label xml:lang="en">textbrick</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="image">
    <rdfs:subClassOf rdf:resource="#content" />
    <rdfs:label xml:lang="en">imagebrick</rdfs:label>
  </owl:Class>
```

```
<owl:Class rdf:ID="document">
  <rdfs:subClassOf rdf:resource="#content" />
  <rdfs:label xml:lang="en">documentbrick</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="service">
  <rdfs:subClassOf rdf:resource="#content" />
  <rdfs:label xml:lang="en">servicebrick</rdfs:label>
</owl:Class>
<owl:DatatypeProperty rdf:ID="operation">
  <rdfs:domain rdf:resource="#service" />
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:Class rdf:ID="serviceParameter">
</owl:Class>
<owl:ObjectProperty rdf:ID="parameter">
  <rdfs:domain rdf:resource="#service" />
  <rdfs:range  rdf:resource="#serviceParameter"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="order">
  <rdfs:domain rdf:resource="#serviceParameter" />
  <rdfs:range  rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hint">
  <rdfs:domain rdf:resource="#serviceParameter" />
  <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="related-content">
  <rdfs:domain rdf:resource="&owl;Thing" />
  <rdfs:range  rdf:resource="#content"/>
</owl:ObjectProperty>

<!--
  Define how domains can be explored
-->
<owl:Class rdf:about="#DomainLookupSpec">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#maxItems" />
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="resourceType">
  <rdfs:domain rdf:resource="#DomainLookupSpec"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="maxItems">
  <rdfs:domain rdf:resource="#DomainLookupSpec"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>

<owl:Class rdf:about="#RelationLookupSpec">
  <rdfs:subClassOf rdf:resource="#DomainLookupSpec" />
</owl:Class>

<owl:DatatypeProperty rdf:ID="type">
  <rdfs:domain rdf:resource="#RelationLookupSpec"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="depth">
  <rdfs:domain rdf:resource="#RelationLookupSpec"/>
```

```
  <rdfs:range rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="relevance">
  <rdfs:domain rdf:resource="#RelationLookupSpec"/>
  <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>

<owl:Class rdf:about="#QuerySpec">
  <rdfs:subClassOf rdf:resource="#DomainLookupSpec" />
</owl:Class>

<owl:DatatypeProperty rdf:ID="query">
  <rdfs:domain rdf:resource="#QuerySpec"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!--
  Adaptation structure definitions
 -->
<owl:Class rdf:about="#Adaptation"/>

<owl:ObjectProperty rdf:ID="preAdaptation">
  <rdfs:domain rdf:resource="#Adaptation"/>
  <rdfs:range rdf:resource="&owl;Thing"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="postAdaptation">
  <rdfs:domain rdf:resource="#Adaptation"/>
  <rdfs:range rdf:resource="&owl;Thing"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="adaptationResourceType">
  <rdfs:domain rdf:resource="#Adaptation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="adaptationMaxItems">
  <rdfs:domain rdf:resource="#Adaptation"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="relevanceThreshold">
  <rdfs:domain rdf:resource="#Adaptation"/>
  <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="itemNamespace">
  <rdfs:domain rdf:resource="#Adaptation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="contextFacet">
  <rdfs:domain rdf:resource="#Adaptation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="domainExploration">
  <rdfs:domain rdf:resource="#Adaptation"/>
  <rdfs:range rdf:resource="#DomainLookupSpec"/>
</owl:ObjectProperty>

<!--
  Define a context rule
  -->
<owl:Class rdf:about="#ContextRule">
```

```
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#factor" />
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#rule" />
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="factor">
  <rdfs:domain rdf:resource="#ContextRule"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="rule">
  <rdfs:domain rdf:resource="#ContextRule"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!--
  Definition of context relations
-->
<owl:Class rdf:about="#ContextRelation">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#relevanceItem" />
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">2</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#influenceValue" />
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="relevanceItem">
  <rdfs:domain rdf:resource="#ContextRelation"/>
  <rdfs:range rdf:resource="&owl;Thing"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="influenceValue">
  <rdfs:domain rdf:resource="#ContextRelation" />
  <rdfs:range  rdf:resource="&xsd;float" />
</owl:DatatypeProperty>

<!--
  Definition of navigation structures
-->
<owl:Class rdf:about="#NavRelation">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#navRelationSrc" />
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#navRelationTarget" />
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
```

```
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#navRelationType" />
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#adaptationProperty" />
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#lookup" />
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="navRelationSrc">
  <rdfs:domain rdf:resource="#NavRelation"/>
  <rdfs:range rdf:resource="&owl;Thing"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="navRelationTarget">
  <rdfs:domain rdf:resource="#NavRelation"/>
  <rdfs:range rdf:resource="&owl;Thing"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="navRelationType">
  <rdfs:domain rdf:resource="#NavRelation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="adaptationProperty">
  <rdfs:domain rdf:resource="#NavRelation"/>
  <rdfs:range rdf:resource="#Adaptation"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="lookup">
  <rdfs:domain rdf:resource="#NavRelation"/>
  <rdfs:range rdf:resource="#DomainLookupSpec"/>
</owl:ObjectProperty>

<!—
  Views: associate view selection information with navigation containers
  —>
<owl:Class rdf:about="#View">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#viewContainer" />
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#contextRule" />
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="viewContainer">
  <rdfs:domain rdf:resource="#View"/>
  <rdfs:range rdf:resource="#NavRelation"/>
```

```
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="contextRule">
    <rdfs:domain rdf:resource="#View"/>
    <rdfs:range rdf:resource="#ContextRule"/>
  </owl:ObjectProperty>

  <!--
    Presentations: associate GUI information with navigation containers
    -->
  <owl:Class rdf:about="#Presentation">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#presentationContainer" />
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#style" />
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#presentationAdaptation" />
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="presentationContainer">
    <rdfs:domain rdf:resource="#Presentation"/>
    <rdfs:range rdf:resource="#NavRelation"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="style">
    <rdfs:domain rdf:resource="#Presentation"/>
    <rdfs:range rdf:resource="#Style"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="presentationAdaptation">
    <rdfs:domain rdf:resource="#Presentation"/>
    <rdfs:range rdf:resource="#Adaptation"/>
  </owl:ObjectProperty>

  <!--
    Styles
    -->
  <owl:Class rdf:about="#Style">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#class" />
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="class">
    <rdfs:domain rdf:resource="#Style"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
</rdf:RDF>
```

Listing B.1: Model elements in OWL syntax

Appendix C

Models for a Prototypical Web Application

This appendix provides the models for the Web application prototype discussed in Section 5.5. These models instantiate the model definitions introduced in Section 3.5 and use the OWL type definitions shown in Appendix B.

Conceptual Model

Listing C.1 shows an extract of the domain ontology used.

```
<?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE rdf:RDF [
   <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
   <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
   <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
   <!ENTITY wise "http://wise.interactivesystems.info/wise#" >
   ]>
<rdf:RDF
   xmlns:owl  ="&owl;"
   xmlns:rdf  = "&rdf;"
   xmlns:rdfs = "&rdfs;"
   xmlns:xsd  = "&xsd;"
   xmlns:wise = "&wise;"
   xmlns      = "http://wise.interactivesystems.info/domain#"
   xml:base   = "http://wise.interactivesystems.info/domain"
  >

   <owl:Ontology rdf:about="">
      <rdfs:comment>Domain ontology for e-commerce case study</rdfs:comment>
      <owl:versionInfo>$Id: domain.owl 688 2006-05-01 17:08:56Z kaltz $</owl:versionInfo>
   </owl:Ontology>

   <!--
     Concepts used: some examples
    -->
   <owl:Class rdf:ID="wear">
      <rdfs:label xml:lang="en">Wear</rdfs:label>
      <rdfs:label xml:lang="de">Kleidung</rdfs:label>
   </owl:Class>

   <owl:Class rdf:ID="shoe">
      <rdfs:subClassOf rdf:resource="#wear" />
      <rdfs:label xml:lang="en">Shoes</rdfs:label>
```

```
    <rdfs:label xml:lang="de">Schuhe</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="sportsshoe">
    <rdfs:subClassOf rdf:resource="#shoe" />
    <rdfs:label xml:lang="en">Sports shoes</rdfs:label>
    <rdfs:label xml:lang="de">Sportschuhe</rdfs:label>
    <wise:related-content rdf:resource="#t0009" />
  </owl:Class>

  <owl:Class rdf:ID="outdoorRunningShoe">
    <rdfs:subClassOf rdf:resource="#sportsshoe" />
    <rdfs:label xml:lang="en">Outdoor running shoes</rdfs:label>
    <rdfs:label xml:lang="de">Laufschuhe für draussen</rdfs:label>
    <wise:related-content rdf:resource="#i0003" />
  </owl:Class>

  <!--
    Instances of concepts: an example
   -->

  <stores rdf:ID="duisburg">
    <rdfs:label xml:lang="en">Duisburg</rdfs:label>
    <rdfs:label xml:lang="de">Duisburg</rdfs:label>
  </stores>

  <!--
    Instances of content resources: some examples
   -->

  <wise:text rdf:ID="t0007">
    <rdfs:label xml:lang="en">Our store located at Forsthausweg 3 is looking forward to
        serving you, Monday to Friday, 8 AM to 8 PM.</rdfs:label>
    <rdfs:label xml:lang="de">Unser Geschäft am Forsthausweg 3 freut sich auf Ihren Besuch
        , von Montag bis Freitag 8 bis 20 Uhr.</rdfs:label>
  </wise:text>

  <wise:text rdf:ID="t0006">
    <rdfs:label xml:lang="en">You may particularly enjoy the following item(s):</
        rdfs:label>
    <rdfs:label xml:lang="de">Speziell für Sie im Angebot:</rdfs:label>
  </wise:text>

  <wise:text rdf:ID="t0008">
    <rdfs:label xml:lang="en">Visit one of our retail outlets. Or perhaps you would like
        to try our auction service?</rdfs:label>
    <rdfs:label xml:lang="de">Besuchen Sie eines unserer Geschäfte. Oder machen Sie doch
        einfach von unserem Auktionsdienst Gebrauch!</rdfs:label>
  </wise:text>

  <wise:text rdf:ID="t0009">
    <rdfs:label xml:lang="en">Appropriate sport shoes are the key to
      avoiding injury. Be sure to choose an appropriate sort, according to
      the type of sport you enjoy.</rdfs:label>
    <rdfs:label xml:lang="de">Geeignete Sportschuhe sind enorm
      wichtig, um Verletzungsgefahr zu minimieren. Wählen Sie Sportschuhe,
      die für die von Ihnen praktizierte Sportart geeignet sind.</rdfs:label>
  </wise:text>

  <wise:text rdf:ID="t0010">
    <rdfs:label xml:lang="en">Special offer of the month: RGG sporting
      socks for a mere 9 $ 95.</rdfs:label>
    <rdfs:label xml:lang="de">Diesen Monat im Spezialangebot:
      RGG Sportlersocken für nur 9 $ 95.</rdfs:label>
  </wise:text>
```

```
  <wise:image rdf:ID="i0003">
    <rdfs:label xml:lang="en">Running outdoors</rdfs:label>
    <rdfs:label xml:lang="de">Laufen in der freien Natur</rdfs:label>
    <wise:source rdf:datatype="&xsd;anyURI">resources/outdoor-running.jpg</wise:source>
  </wise:image>

  <wise:document rdf:ID="d0002">
    <rdfs:label xml:lang="en">Download the entire product catalog</rdfs:label>
    <rdfs:label xml:lang="de">Hier können Sie den kompletten Produktkatalog
      herunterladen</rdfs:label>
    <wise:source rdf:datatype="&xsd;anyURI">resources/product_catalog.pdf</wise:source>
  </wise:document>

  <wise:service rdf:ID="s0001">
    <rdfs:label xml:lang="en">Auction service</rdfs:label>
    <rdfs:label xml:lang="de">Auktionsdienst</rdfs:label>
    <wise:source rdf:datatype="&xsd;anyURI">resources/AuctionService.wsdl</wise:source>
    <wise:operation>search</wise:operation>
    <wise:parameter rdf:resource="#sp001" />
    <wise:parameter rdf:resource="#sp002" />
    <wise:parameter rdf:resource="#sp003" />
  </wise:service>

  <wise:serviceParameter rdf:ID="sp001">
    <wise:hint>http://wise.interactivesystems.info/domain#wear</wise:hint>
    <wise:order>1</wise:order>
    <rdfs:label xml:lang="en">Category</rdfs:label>
    <rdfs:label xml:lang="de">Kategorie</rdfs:label>
  </wise:serviceParameter>

  <wise:serviceParameter rdf:ID="sp002">
    <wise:hint>http://wise.interactivesystems.info/context/time#Season</wise:hint>
    <wise:order>2</wise:order>
    <rdfs:label xml:lang="en">Season</rdfs:label>
    <rdfs:label xml:lang="de">Jahreszeit</rdfs:label>
  </wise:serviceParameter>

  <wise:serviceParameter rdf:ID="sp003">
    <wise:hint>size</wise:hint>
    <wise:order>3</wise:order>
    <rdfs:label xml:lang="en">Size</rdfs:label>
    <rdfs:label xml:lang="de">Größe</rdfs:label>
  </wise:serviceParameter>

</rdf:RDF>
```

Listing C.1: Conceptual model for the case study

Navigation Model

Listing C.2 shows an extract of the navigation model used.

```
<?xml version="1.0" encoding="ISO-8859-15"?>

<!DOCTYPE rdf:RDF [
  <!ENTITY xsd      "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY owl      "http://www.w3.org/2002/07/owl#" >
  <!ENTITY domain   "http://wise.interactivesystems.info/domain#" >
  <!ENTITY wise     "http://wise.interactivesystems.info/wise#" >
  ]>

<rdf:RDF
```

```
xmlns:xsd ="&xsd;"
xmlns:owl ="&owl;"
xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:wise = "&wise;"
xmlns="http://wise.interactivesystems.info/navigation#"
xml:base="http://wise.interactivesystems.info/navigation"
>

<owl:Ontology rdf:about="">
   <rdfs:comment>Navigation Model</rdfs:comment>
   <owl:versionInfo>$Id: navModel.owl 688 2006-05-01 17:08:56Z kaltz $</owl:versionInfo>
</owl:Ontology>

<!--
  Examples of fixed navigation relations
 -->

<wise:NavRelation rdf:ID="r1">
    <wise:navRelationSrc rdf:resource="&domain;home" />
    <wise:navRelationTarget rdf:resource="&domain;tips" />
    <wise:navRelationType>link</wise:navRelationType>
</wise:NavRelation>

<wise:NavRelation rdf:ID="r2">
    <wise:navRelationSrc rdf:resource="&domain;home" />
    <wise:navRelationTarget rdf:resource="&domain;wear" />
    <wise:navRelationType>link</wise:navRelationType>
</wise:NavRelation>

<wise:NavRelation rdf:ID="r5">
    <wise:navRelationSrc rdf:resource="&domain;home" />
    <wise:navRelationTarget rdf:resource="&domain;t0001" />
    <wise:navRelationType>content</wise:navRelationType>
</wise:NavRelation>

<wise:NavRelation rdf:ID="r6">
    <wise:navRelationSrc rdf:resource="&domain;home" />
    <wise:navRelationTarget rdf:resource="&domain;t0002" />
    <wise:navRelationType>content</wise:navRelationType>
</wise:NavRelation>

<wise:NavRelation rdf:ID="r8b">
    <wise:navRelationSrc rdf:resource="&domain;wear" />
    <wise:navRelationTarget rdf:resource="&domain;d0002" />
    <wise:navRelationType>content</wise:navRelationType>
</wise:NavRelation>

<!--
  Example navigation relations via dynamic ontology lookup:
  association of sub-links or content pieces;
  lookup specification via type or via freehand query
 -->

<wise:NavRelation rdf:ID="r9">
    <wise:navRelationSrc rdf:resource="&domain;shoe" />
    <wise:lookup rdf:resource="rLookup1" />
    <wise:navRelationType>link</wise:navRelationType>
</wise:NavRelation>

<wise:RelationLookupSpec rdf:ID="rLookup1">
    <wise:type>subclass</wise:type>
    <wise:depth>1</wise:depth>
</wise:RelationLookupSpec>

<wise:NavRelation rdf:ID="r18b">
```

```
    <wise:navRelationSrc rdf:resource="&domain;sportsshoe" />
    <wise:lookup rdf:resource="rLookup2" />
    <wise:navRelationType>content</wise:navRelationType>
</wise:NavRelation>

<wise:RelationLookupSpec rdf:ID="rLookup2">
    <wise:type>&wise;related-content</wise:type>
    <wise:depth>1</wise:depth>
</wise:RelationLookupSpec>

<wise:NavRelation rdf:ID="r21">
    <wise:navRelationSrc rdf:resource="&domain;special_offers" />
    <wise:lookup rdf:resource="q1" />
    <wise:navRelationType>content</wise:navRelationType>
</wise:NavRelation>

<wise:QuerySpec rdf:ID="q1">
    <wise:query>
        prefix wise: &lt;http://wise.interactivesystems.info/wise#&gt;
        prefix domain: &lt;http://wise.interactivesystems.info/domain#&gt;
        select ?uri where {domain:special_offers wise:related-content ?uri .}
    </wise:query>
</wise:QuerySpec>

<!--
  Adaptation examples
  -->
<wise:NavRelation rdf:ID="ar1">
    <wise:navRelationSrc rdf:resource="&domain;wear" />
    <wise:navRelationType>content</wise:navRelationType>
    <wise:adaptationProperty rdf:resource="a1"/>
</wise:NavRelation>

<wise:Adaptation rdf:ID="a1">
    <wise:adaptationResourceType>any</wise:adaptationResourceType>
    <wise:itemNamespace>&domain;</wise:itemNamespace>
    <wise:adaptationMaxItems>3</wise:adaptationMaxItems>
    <wise:relevanceThreshold>0.1</wise:relevanceThreshold>
    <wise:preAdaptation rdf:resource="&domain;t0006" />
    <wise:domainExploration rdf:resource="d0"/>
</wise:Adaptation>

<wise:RelationLookupSpec rdf:ID="d0">
    <wise:type>http://wise.interactivesystems.info/wise#related-content</wise:type>
    <wise:depth>1</wise:depth>
    <wise:relevance>90</wise:relevance>
</wise:RelationLookupSpec>

<wise:NavRelation rdf:ID="ar2">
    <wise:navRelationSrc rdf:resource="&domain;tips" />
    <wise:navRelationType>link</wise:navRelationType>
    <wise:adaptationProperty rdf:resource="a2"/>
</wise:NavRelation>

<wise:Adaptation rdf:ID="a2">
    <wise:adaptationResourceType>any</wise:adaptationResourceType>
    <wise:itemNamespace>&domain;</wise:itemNamespace>
    <wise:adaptationMaxItems>5</wise:adaptationMaxItems>
    <wise:relevanceThreshold>0.1</wise:relevanceThreshold>
    <wise:domainExploration rdf:resource="d1"/>
    <wise:domainExploration rdf:resource="d2"/>
</wise:Adaptation>

<wise:RelationLookupSpec rdf:ID="d1">
    <wise:type>superclass</wise:type>
    <wise:depth>2</wise:depth>
```

```
        <wise:relevance>90</wise:relevance>
    </wise:RelationLookupSpec>

    <wise:RelationLookupSpec rdf:ID="d2">
        <wise:type>subclass</wise:type>
        <wise:depth>2</wise:depth>
        <wise:relevance>95</wise:relevance>
    </wise:RelationLookupSpec>

</rdf:RDF>
```

Listing C.2: Navigation model for the case study

View Model

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl      "http://www.w3.org/2002/07/owl#" >
    <!ENTITY view     "http://wise.interactivesystems.info/view" >
    <!ENTITY wise     "http://wise.interactivesystems.info/wise#" >
    <!ENTITY nav      "http://wise.interactivesystems.info/navigation#" >
    <!ENTITY device   "http://wise.interactivesystems.info/context/device#">
]>

<rdf:RDF
    xmlns:owl ="&owl;"
    xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:wise="&wise;"
    xmlns:nav="&nav;"
    xmlns="&view;#"
    xml:base="&view;"
>
    <owl:Ontology rdf:about="">
        <rdfs:comment>View Model</rdfs:comment>
        <owl:versionInfo>$Id: view.owl 620 2006-03-25 15:30:48Z kaltz $</owl:versionInfo>
    </owl:Ontology>

    <wise:View rdf:ID="v1">
        <wise:viewContainer rdf:resource="&nav;r8b"/>
        <wise:contextRule rdf:resource="#cRule1"/>
    </wise:View>

    <wise:ContextRule rdf:ID="cRule1">
        <wise:factor>&device;desktop</wise:factor>
        <wise:rule>true</wise:rule>
    </wise:ContextRule>

</rdf:RDF>
```

Listing C.3: View model for the case study

Presentation Model

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl          "http://www.w3.org/2002/07/owl#" >
```

```
    <!ENTITY presentation "http://wise.interactivesystems.info/presentation#" >
    <!ENTITY wise         "http://wise.interactivesystems.info/wise#" >
    <!ENTITY nav          "http://wise.interactivesystems.info/navigation#" >
]>

<rdf:RDF
    xmlns:owl  = "&owl;"
    xmlns:rdf  = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
    xmlns:wise = "&wise;"
    xmlns:nav  = "&nav;"
    xmlns      = "&presentation;"
    xml:base   = "&presentation;"
>
    <owl:Ontology rdf:about="">
       <rdfs:comment>Presentation Model</rdfs:comment>
       <owl:versionInfo>$Id: presentation.owl 620 2006-03-25 15:30:48Z kaltz $</
            owl:versionInfo>
    </owl:Ontology>

    <!--
      Instances of presentations
      -->
    <wise:Presentation rdf:ID="p1">
       <wise:presentationContainer rdf:resource="&nav;r20a" />
       <wise:style rdf:resource="#style2" />
       <wise:presentationAdaptation rdf:resource="#a1" />
    </wise:Presentation>

    <!--
      Adaptation property instances
      -->
    <wise:Adaptation rdf:ID="a1">
        <wise:itemNamespace>&presentation;</wise:itemNamespace>
        <wise:adaptationMaxItems>1</wise:adaptationMaxItems>
        <wise:relevanceThreshold>0.1</wise:relevanceThreshold>
    </wise:Adaptation>

    <!--
      Instances of style
      -->
    <wise:Style rdf:ID="style1">
       <wise:class>sportyStore</wise:class>
    </wise:Style>

    <wise:Style rdf:ID="style2">
       <wise:class>conservativeStore</wise:class>
    </wise:Style>

</rdf:RDF>
```

Listing C.4: Presentation model for the case study

Context Relations

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY xsd       "http://www.w3.org/2001/XMLSchema#">
    <!ENTITY owl       "http://www.w3.org/2002/07/owl#">
    <!ENTITY wise      "http://wise.interactivesystems.info/wise#" >
    <!ENTITY relevance "http://wise.interactivesystems.info/relevance#">
    <!ENTITY domain    "http://wise.interactivesystems.info/domain#">
```

```
    <!ENTITY time          "http://wise.interactivesystems.info/context/time#">
    <!ENTITY user          "http://wise.interactivesystems.info/context/user#">
    <!ENTITY device        "http://wise.interactivesystems.info/context/device#">
    <!ENTITY location      "http://wise.interactivesystems.info/context/location#">
    <!ENTITY view          "http://wise.interactivesystems.info/view#">
    <!ENTITY presentation "http://wise.interactivesystems.info/presentation#">
    ]>

<rdf:RDF
     xmlns:owl        ="http://www.w3.org/2002/07/owl#"
     xmlns:rdf        ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs       ="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd        ="http://www.w3.org/2001/XMLSchema#"
     xmlns:wise       = "&wise;"
     xmlns            = "&relevance;"
     xmlns:relevance  = "&relevance;"
     xml:base         = "&relevance;"

     xmlns:domain       = "&domain;"
     xmlns:time         = "&time;"
     xmlns:user         = "&user;"
     xmlns:device       = "&device;"
     xmlns:location     = "&location;"
     xmlns:view         = "&view;"
     xmlns:presentation = "&presentation;"
    >
    <owl:Ontology rdf:about="">
       <rdfs:comment>Context Relations Ontology</rdfs:comment>
       <owl:versionInfo>$Id: contextrelations.owl 589 2006-03-19 18:55:24Z kaltz $</
            owl:versionInfo>
       <rdfs:label>Context Relations Ontology</rdfs:label>
    </owl:Ontology>

    <wise:ContextRelation rdf:ID="re01">
       <wise:relevanceItem rdf:resource="&domain;outdoorRunningShoe" />
       <wise:relevanceItem rdf:resource="&time;summer" />
       <wise:influenceValue rdf:datatype="&xsd;float">1.0</wise:influenceValue>
    </wise:ContextRelation>

    <wise:ContextRelation rdf:ID="re02">
       <wise:relevanceItem rdf:resource="&domain;outdoorRunningShoe" />
       <wise:relevanceItem rdf:resource="&time;summer" />
       <wise:relevanceItem rdf:resource="&user;Athlete" />
       <wise:influenceValue rdf:datatype="&xsd;float">1.0</wise:influenceValue>
    </wise:ContextRelation>

    <wise:ContextRelation rdf:ID="re03">
       <wise:relevanceItem rdf:resource="&user;Sportsman" />
       <wise:relevanceItem rdf:resource="&presentation;style1" />
       <wise:influenceValue rdf:datatype="&xsd;float">1.0</wise:influenceValue>
    </wise:ContextRelation>

    <wise: rdf:ID="re04">
       <wise:relevanceItem rdf:resource="&domain;skiClothing" />
       <wise:relevanceItem rdf:resource="&time;autumn" />
       <wise:influenceValue rdf:datatype="&xsd;float">1.0</wise:influenceValue>
    </wise:ContextRelation>

</rdf:RDF>
```

Listing C.5: Context relations model for the case study

Appendix D

CATWALK Framework UML Diagrams

This appendix contains supplementary UML diagrams to document the CATWALK framework, as described in Chapter 5.

Figure D.1 provides the class diagram containing interface and default implementations for the *ContextExtractor* component.

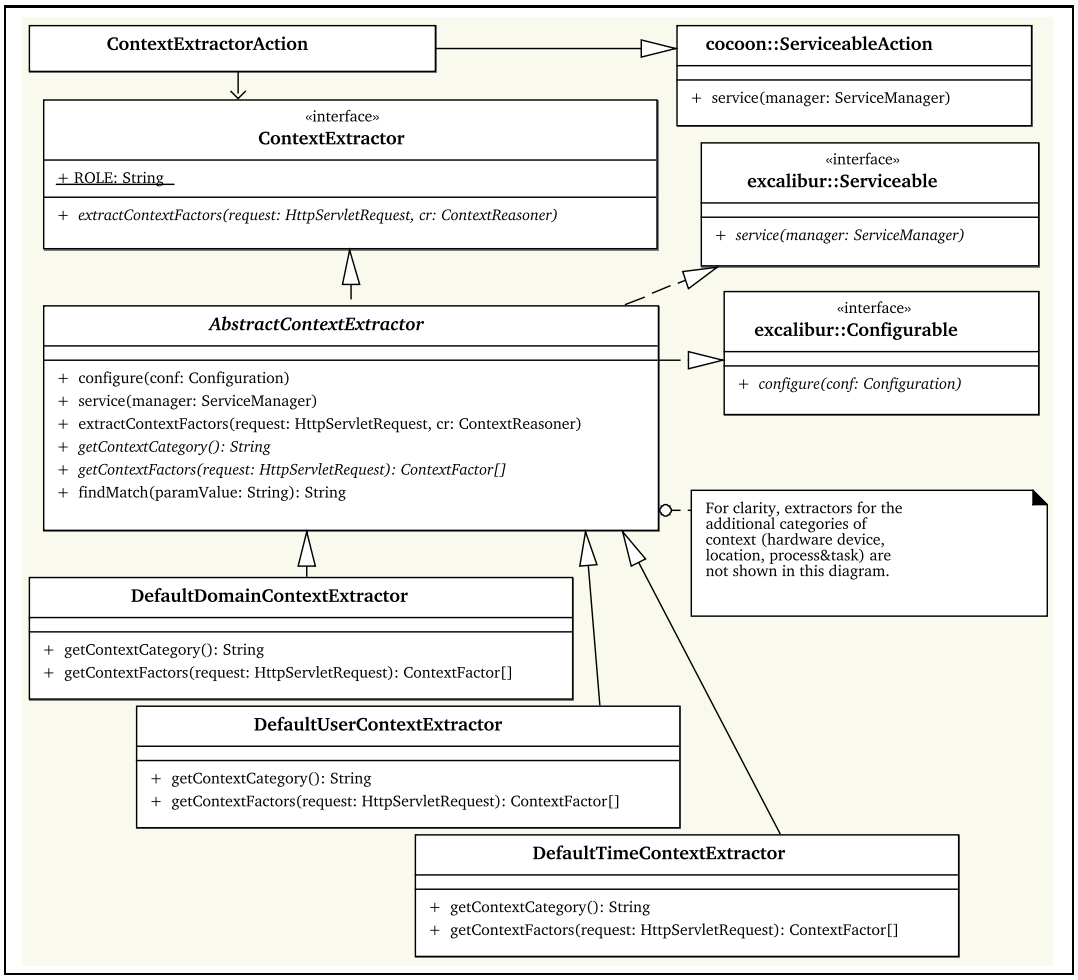

Figure D.1: *ContextExtractor* class diagram

Figure D.2 describes how the framework sets up an object-oriented representation of the context model (in the figure, the instance `cmodel` of the class `ContextModel`) by using Excalibur's source resolving mechanism. This mechanism in turn allows for accessing a, possibly remote, location of model sources.

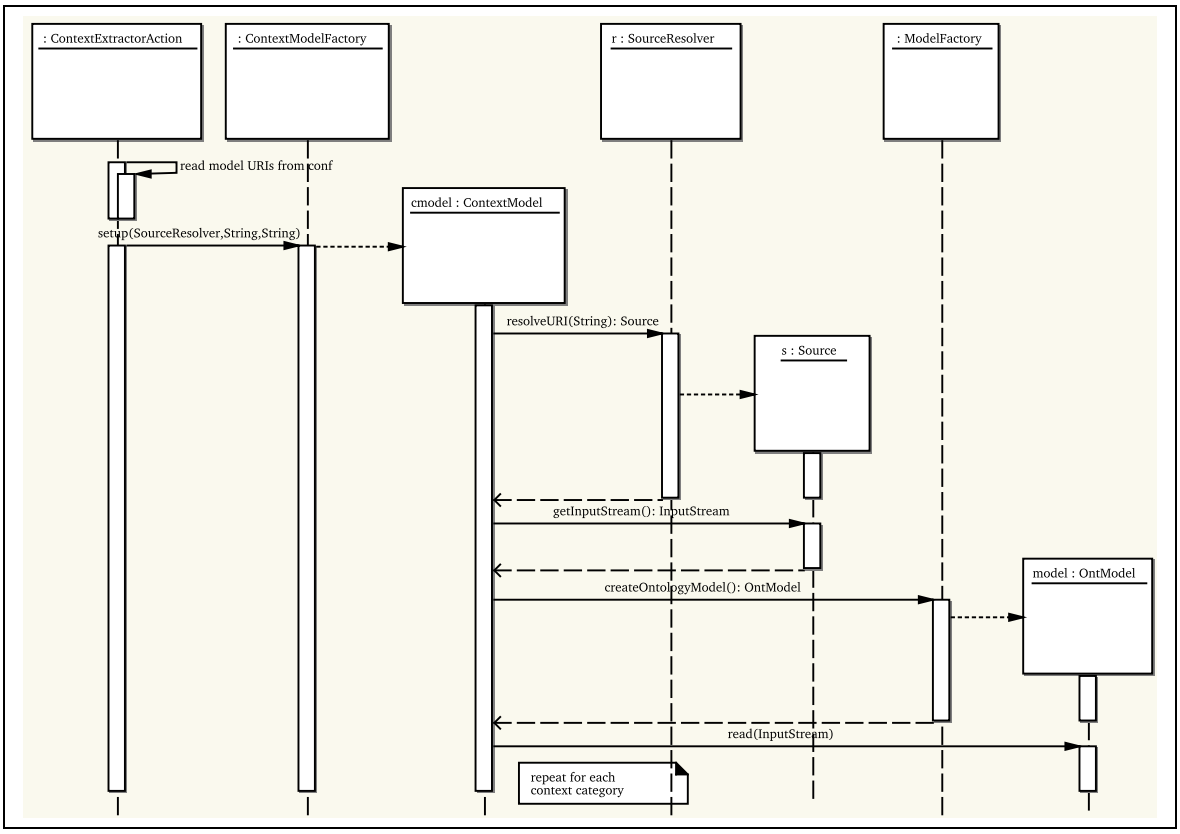

Figure D.2: Context model setup sequence diagram

Figure D.3 shows a class diagram for the components and classes used in supporting the feature of context simulation. The *generators* provide an XML representation of current application information. The XML-serialized representation is usable for GUI generation, and for debugging during development, for example via logging statements. The `ContextStateGenerator` outputs context state; the `ContextHistoryGenerator` outputs past states of context. The `ContextModelGenerator` can be used to output available context factors (see Section 5.2.3) in a serialized form.

The integration of the simulation generator components in a Cocoon pipeline is illustrated in the pipeline processing diagrams of Figure D.4. The upper pipeline creates the GUI for the context simulator. The action registered as `contextinfo`, implemented by the `ContextInfoAction` class, is called prior to each request for the context simulation GUI, to check if context and application models are to be reloaded. Output from the generators as described above is then aggregated, and an XSLT stylesheet applied to generate a user interface.

When the user chooses to activate or deactivate a factor, the lower pipeline is processed, resulting in the action registered as `contextsimulator` being executed. This action is implemented by the `ContextSimulatorAction` Java class, whose processing steps are illustrated in the following.

The sequence diagram in Figure D.5 documents the processing steps triggered by an execution of the `contextsimulator` action. The method `applySimulationCommands` is called to set the context

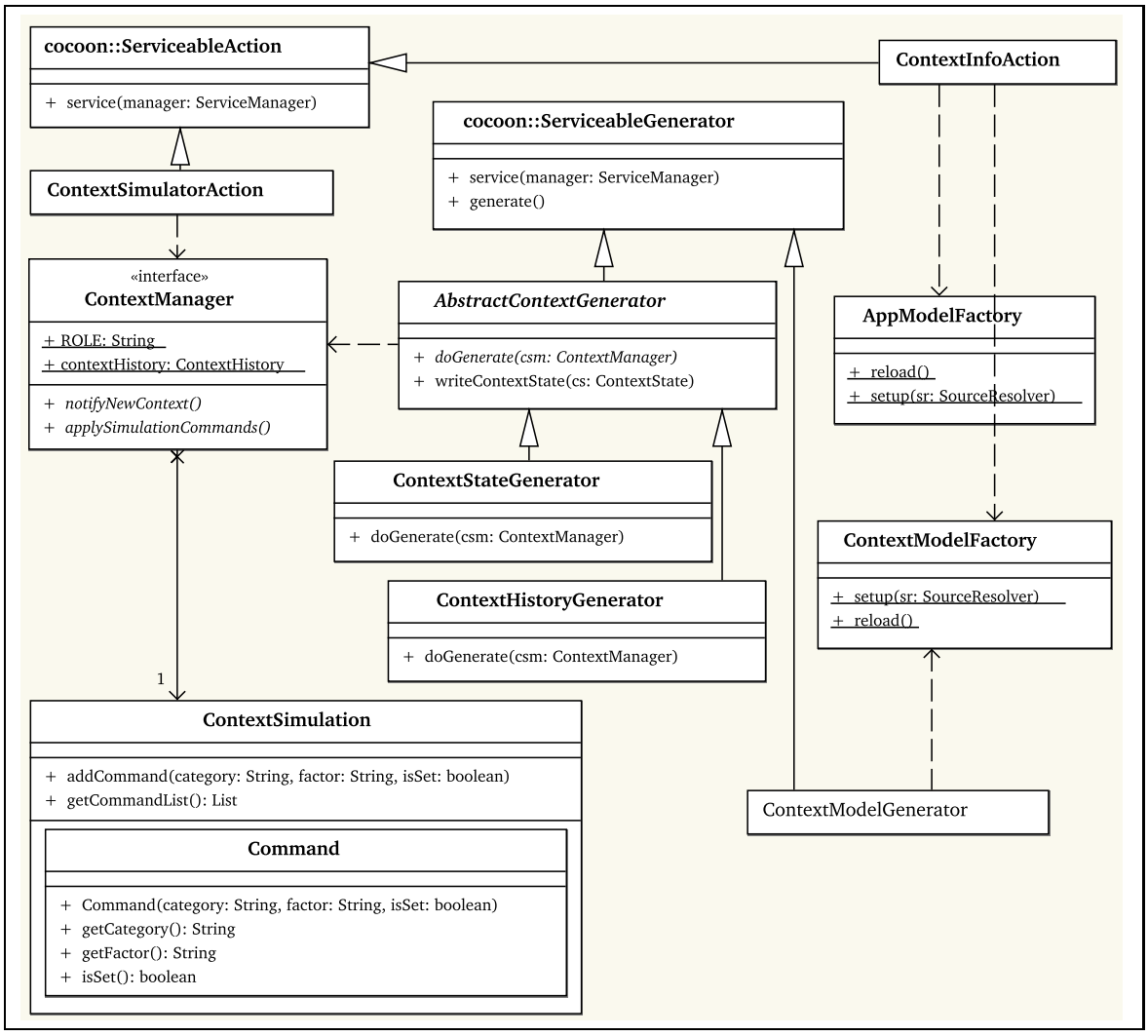

Figure D.3: Context simulation class diagram

factors the user wishes to simulate. This processing step is also called at each request within the actual Web application, to ensure that any existing simulation commands are processed after context sensing mechanisms. For simplicity, the sequence diagram refers to a *ContextManager*, which is in fact a component interface (see Section 5.2.1); in the actual system, a concrete component registered for this instance is retrieved by Excalibur.

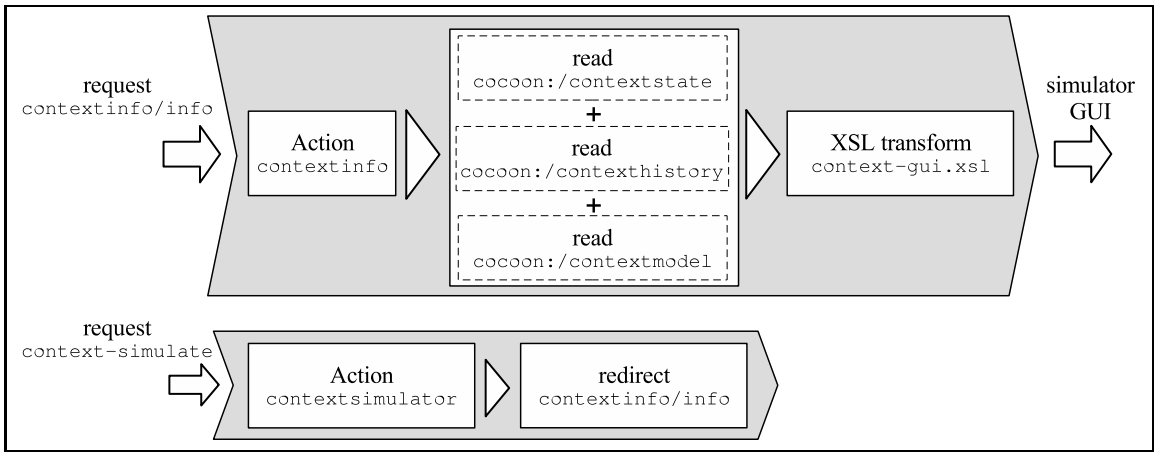

Figure D.4: Context simulation pipelines

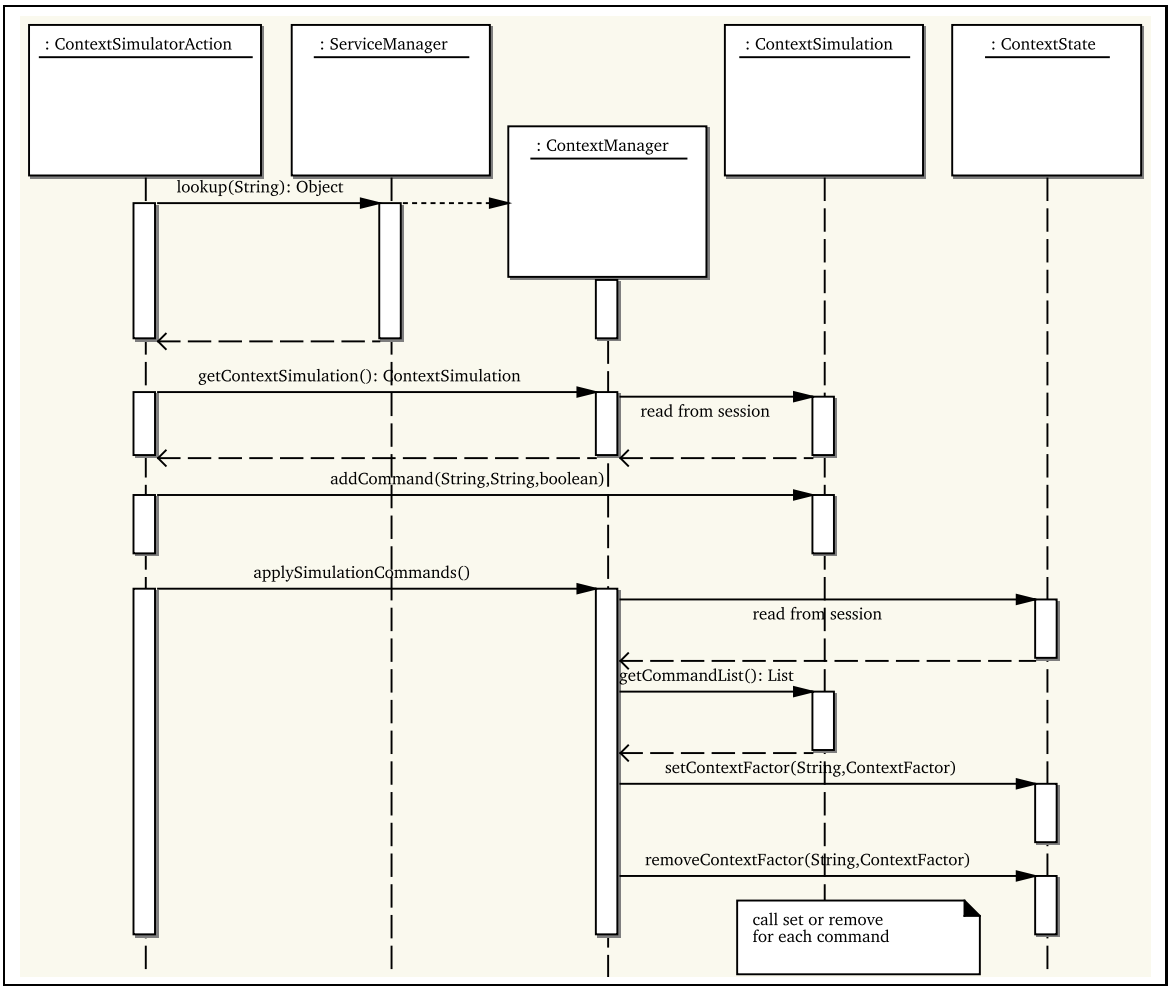

Figure D.5: Context simulation processing sequence diagram

List of Figures

1.1 Chapter overview . . . 21

2.1 A three-tier architecture based on Cocoon . . . 36

3.1 A context-aware e-commerce Web application . . . 47
3.2 Context factor categories . . . 50
3.3 Context knowledge and relations . . . 56
3.4 Context model for an e-commerce scenario . . . 57
3.5 Context model for an industry maintenance scenario . . . 58
3.6 Context modeling augmenting the WISE methodology . . . 60
3.7 Content block types . . . 63
3.8 Representation of a context relation . . . 64
3.9 Context relation example . . . 64
3.10 Ontology lookup specification . . . 64
3.11 Adaptation specification . . . 65
3.12 Navigation model entry definition . . . 67
3.13 View and presentation model definition . . . 67
3.14 Editing context relations using the MatrixBrowser . . . 68
3.15 ContextBrowser: multi-dimensional context relations . . . 69

4.1 Operational environment of the context-aware architecture . . . 72
4.2 Composite structure diagram of the context-aware architecture . . . 73
4.3 Types of context-relevant resources . . . 75
4.4 Cocoon pipeline example . . . 79
4.5 Cocoon component architecture . . . 81
4.6 Context-leveraging stack . . . 84
4.7 Example of top-level user interface structures and stylesheets . . . 92
4.8 Principles for a SoC-based Web software architecture for context (WSAC) . . . 93

5.1 Context framework component architecture . . . 96
5.2 *ContextManager* class diagram . . . 97
5.3 Request context extraction and notification sequence diagram . . . 99
5.4 *ContextReasoner* component class diagram . . . 100
5.5 Context sensing activity regarding context state . . . 101
5.6 Model representation class diagram . . . 102
5.7 Ontology lookup class diagram . . . 103
5.8 Ontology exploration class diagram . . . 104

5.9 *AdaptationEngine* interface 105
5.10 Adaptive transformers: class diagram 108
5.11 Navigation tree example 108
5.12 Pipeline processing for navigation structure generation 109
5.13 Pipeline processing for content generation 111
5.14 Processing flow for response generation to a typical user request 114
5.15 Case study: application screenshot 116
5.16 Case study: model extract 117
5.17 Navigation adaptation demonstration 117
5.18 Content adaptation demonstration 118
5.19 Service adaptation demonstration 1 118
5.20 Service adaptation demonstration 2 119
5.21 Presentation adaptation demonstration - View 120
5.22 Presentation adaptation demonstration - GUI attributes 120
5.23 Context simulation screenshot 121

D.1 *ContextExtractor* class diagram 145
D.2 Context model setup sequence diagram 146
D.3 Context simulation class diagram 147
D.4 Context simulation pipelines 148
D.5 Context simulation processing sequence diagram 148

List of Tables

1.1 Network-centric applications in a professional environment 15

2.1 Applications using context-awareness 24
2.2 Architectural approaches in research regarding adaptivity 29

3.1 Adaptation effects in a Web application 45
3.2 Context characteristics 51

4.1 Content-driven vs. technology-driven representation 76
4.2 General architectural approaches for adaptive systems 77
4.3 Component type overview 86

6.1 Accounting for, and using context: summary 127

A.1 Standard authorities 129
A.2 Standards 129

List of Listings

4.1 Cocoon pipeline configuration example 79
4.2 GUI pattern and XSLT templating example 92
5.1 Context extractor configuration 98
5.2 Sitemap configuration for model access 103
5.3 Navigation generation input structure definition 110
5.4 Navigation generation output structure definition 110
5.5 Content generation structure definition 112
5.6 Service generation structure definition 112
B.1 Model elements in OWL syntax 131
C.1 Conceptual model for the case study 137
C.2 Navigation model for the case study 139
C.3 View model for the case study 142
C.4 Presentation model for the case study 142
C.5 Context relations model for the case study 143

Bibliography

Rakesh Agrawal, Jr. Roberto J. Bayardo, Daniel Gruhl, and Spiros Papadimitriou. Vinci: a service-oriented architecture for rapid development of web applications. In *WWW '01: Proceedings of the tenth international conference on World Wide Web*, pages 355–365. ACM Press, 2001.

Grigoris Antoniou and Frank van Harmelen. Web Ontology Language. In *Handbook on Ontologies*. Springer-Verlag, 2004.

Dhouha Ayed, Chantal Taconet, Nawel Sabri, and Guy Bernard. Context-aware distributed deployment of component-based applications. In Robert Meersman, Zahir Tari, and Angelo Corsaro, editors, *OTM Workshops*, volume 3292 of *Lecture Notes in Computer Science*, pages 36–37. Springer, 2004.

B. Badrinath, Armando Fox, Leonard Kleinrock, Gerald Popek, Peter Reiher, and M. Satyanarayanan. A conceptual framework for network and client adaptation. *Mobile Networks and Applications*, 5 (4):221–231, 2000.

Sean Baker. Web Services and CORBA. In *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002*, pages 618–632, London, UK, 2002. Springer-Verlag.

Stefania Bandini, Flavio De Paoli, Sara Manzoni, and Paolo Mereghetti. A support system to COTS-based software development for business services. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 307–314. ACM Press, 2002.

Luciano Baresi, Franca Garzotto, and Paolo Paolini. From web sites to web applications: New issues for conceptual modeling. In *ER '00: Proceedings of the Workshops on Conceptual Modeling Approaches for E-Business and The World Wide Web and Conceptual Modeling*, pages 89–100, London, UK, 2000. Springer-Verlag.

Chris Barry and Michael Lang. A survey of multimedia and web development techniques and methodology usage. *IEEE MultiMedia*, 8(2):52–60, 2001.

Christian Becker and Daniela Nicklas. Where do spatial context-models end and where do ontologies start? A proposal of a combined approach. In *Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management in conjunction with UbiComp 2004*, 2004.

Michael Beigl, Albert Krohn, Tobias Zimmer, Christian Decker, and Philip Robinson. AwareCon: Situation Aware Context Communication. In *Proceedings of UbiComp 2003: Ubiquitous Computing*, pages 132–139, Berlin, 2003. Springer.

Fernando Bellas, Daniel Fernandez, and Abel Muino. A flexible framework for engineering "my" portals. In *Proceedings of the 13th international conference on World Wide Web*, pages 234–243. ACM Press, 2004.

Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

Howard Block, Rob Castle, and David Hritz. *Creating Web Portals with BEA WebLogic*. Apress, 2003.

Stefano Bocconi and Frank Nack. Vox populi: Automatic generation of biased video sequences. In *Proceedings of the 1st ACM workshop on Story representation, mechanism and context*, pages 9–16. ACM Press, 2004.

David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture. Working Group Note, http://www.w3.org/TR/ws-arch/, W3C, 2004.

Jan Bosch. Design Patterns as Language Constructs. *Journal of Object-Oriented Programming*, 11 (2):18–32, 1998.

Henning Böttger, Anders Møller, and Michael I. Schwartzbach. Contracts for cooperation between Web service programmers and HTML designers. *Journal of Web Engineering*, 5(1):65–89, 2006.

Craig Boyle and Swee Hor Teh. Multimedia intelligent documentation: Metadoc v. In *SIGDOC '93: Proceedings of the 11th annual international conference on Systems documentation*, pages 21–27, New York, NY, USA, 1993. ACM Press.

Claus Brabrand, Anders Møller, Anders Sandholm, and Michael I. Schwartzbach. A runtime system for interactive Web services. *Computer Networks*, 31(11-16):1391–1401, May 1999.

Martin Bravenboer and Eelco Visser. Concrete syntax for objects: domain-specific language embedding and assimilation without restrictions. In *Proceedings of the 19th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications*, pages 365–383. ACM Press, 2004.

Patrick Brézillon. Context in problem solving: A survey. *The Knowledge Engineering Review*, 14(1): 47–80, 1999.

Patrick Brézillon. Using Context for Supporting Users Efficiently. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 5*, page 127.3. IEEE Computer Society, 2003.

Patrick Brézillon. A Context Approach of Social Networks. In *Proceedings of the KI-2004 International Workshop on Modelling and Retrieval of Context, Vol-114*. CEUR, 2004.

Patrick Brézillon, Marcos Borges, Jose Pino, and Jean-Charles Pomerol. Context-awareness in group work: three case studies. In R. Meredith, G. Shanks, D. Arnott, and S. Carlsson, editors, *IFIP International Conference on Decision Support Systems (DSS-2004)*, pages 115–124. Monash University, Australia, 2004.

Peter Brusilovsky, Alfred Kobsa, and Julita Vassileva. *Adaptive Hypertext and Hypermedia*. Kluwer Academic Publishers, 1998.

Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

Jenna Burrell, Paul Treadwell, and Geri K. Gay. Designing for context: usability in a ubiquitous environment. In *Proceedings on the 2000 conference on Universal Usability*, pages 80–84. ACM Press, 2000.

Christoph Bussler, Dieter Fensel, and Alexander Maedche. A conceptual architecture for semantic web enabled web services. *SIGMOD Record*, 31(4):24–29, 2002.

Mario Cannataro, Alfredo Cuzzocrea, and Andrea Pugliese. XAHM: an adaptive hypermedia model based on XML. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 627–634, New York, NY, USA, 2002. ACM Press.

Mario Cannataro and Andrea Pugliese. A Flexible Architecture for Adaptive Hypermedia Systems. In *IJCAI's Workshop on Intelligent Techniques for Web Personalization*, 2001.

Leslie Carr, Wendy Hall, Sean Bechhofer, and Carole Goble. Conceptual linking: ontology-based open hypermedia. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 334–342, New York, NY, USA, 2001. ACM Press.

Ricardo Carreira, Jaime M. Crato, Daniel Gonçalves, and Joaquim A. Jorge. Evaluating adaptive user profiles for news classification. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*, pages 206–212. ACM Press, 2004.

Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference - Alternate track papers & posters*, pages 74–83, New York, NY, USA, 2004. ACM Press.

Stefano Ceri, Florian Daniel, and Maristella Matera. Extending webml for modeling multi-channel contextaware web applications. In *Proceedings of WISE - MMIS'03 Workshop*, pages 615–626. IEEE Computer Society, 2003.

Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6): 137–157, 2000.

Humberto Cervantes and Richard S. Hall. Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 614–623, Washington, DC, USA, 2004. IEEE Computer Society.

Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000. URL `citeseer.ist.psu.edu/chen00survey.html`.

Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 18(3):197–207, May 2004.

Peter Chen. The entity-relationship model–toward a unified view of data. *ACM Transactions on Database Systems*, Vol. 1, No. 1:9–36, 1976.

Keith Cheverst, Nigel Davies, Keith Mitchell, Adrian Friday, and Christos Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24. ACM Press, 2000.

Dickson K. W. Chiu, Danny Kok, Alex K. C. Lee, and S. C. Cheung. Integrating Heterogeneous Web Services with WebXcript. In *COMPSAC '03: Proceedings of the 27th Annual International Conference on Computer Software and Applications*, page 272. IEEE Computer Society, 2003.

Tim Clerckx, Frederik Winters, and Karin Coninx. Tool support for designing context-sensitive user interfaces using a model-based approach. In *TAMODIA '05: Proceedings of the 4th international workshop on Task models and diagrams*, pages 11–18, New York, NY, USA, 2005. ACM Press.

Jim Conallen. *Building Web applications with UML (2nd Edition)*. Addison-Wesley, Reading, Mass., 2002.

Jeff Conklin. Hypertext: an introduction and survey. *Computer*, 20(9):17–41, 1987.

K.W. Copeland and C.J. Hwang. Third generation web applications, full service intranets, edi - the fully integrated business model for electronic commerce. In *Proceedings of the 7th Annual Internet Society Networking Conference*, 1997.

G. Costagliola, F. Ferrucci, and R. Francese. Web engineering: Models and methodologies for the design of hypermedia applications. *Handbook of Software Engineering & Knowledge Engineering*, 8:181–199, 2002.

Michel Crampes and Sylvie Ranwez. Ontology-supported and ontology-driven conceptual navigation on the world wide web. In *HYPERTEXT '00: Proceedings of the eleventh ACM on Hypertext and hypermedia*, pages 191–199. ACM Press, 2000.

Tony Culshaw. Enterprise Application Integration using Apache Cocoon 2.1. Website, 2003. `http://www.xml.com/pub/a/2003/11/12/cocoon-eai.html`. Last visited May 20, 2006.

Krzysztof Czarnecki and Ulrich Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

Ravi Darbhamulla and Pamela Lawhead. Paving the way towards an efficient learning management system. In *Proceedings of the 42nd annual Southeast regional conference*, pages 428–433. ACM Press, 2004.

Andrea R. de Andrade, Ethan V. Munson, and Mariada G. Pimentel. A document-based approach to the generation of web applications. In *Proceedings of the 2004 ACM symposium on Document engineering*, pages 45–47. ACM Press, 2004.

O. M. F. De Troyer and C. J. Leune. WSDM: a user centered design method for Web sites. *Computer Networks and ISDN Systems*, 30(1-7):85–94, 1998.

Yogesh Deshpande and San Murugesan. Summary of the second ICSE workshop on web engineering. *SIGSOFT Software Engineering Notes*, 26(1):76–77, 2001.

Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human Computer Interaction*, 16(2-4), 2001.

Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. a cappella: programming by demonstration of context-aware applications. In *CHI '04: Proceedings of the 2004 conference on Human factors in computing systems*, pages 33–40. ACM Press, 2004.

Alan Dix, Janet Finlay, Gregory Abowd, and Russell Beale. *Human-computer interaction, third edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003.

Alan Dix, Tom Rodden, Nigel Davies, Jonathan Trevor, Adrian Friday, and Kevin Palfreyman. Exploiting space and location as a design framework for interactive mobile systems. *Transactions on Computer-Human Interaction*, 7(3):285–321, 2000.

Mark Doernhoefer. Surfing the net for software engineering notes. *SIGSOFT Software Engineering Notes*, 29(3):15–24, 2004.

Paul Dourish. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8(1):19–30, 2004.

Reiner Dumke, Mathias Lother, Cornelius Wille, and Fritz Zbrog. *Web Engineering*. Pearson Studium, 2003.

Andreas Eberhart and Stefan Fischer. *Web Services*. Hanser Fachbuchverlag, 2003.

Christopher C. Ellsworth, Jr. James B. Fenwick, and Barry L. Kurtz. The Quiver system. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 205–209. ACM Press, 2004.

Thomas Erl. *Service-Oriented Architecture : Concepts, Technology, and Design*. Prentice Hall, 2005.

Steven Feiner, Blair Macintyre, and Dorée Seligmann. Knowledge-based augmented reality. *Communications of the ACM*, 36(7):53–62, 1993.

Zoltán Fiala, Michael Hinz, Geert-Jan Houben, and Flavius Frasincar. Design and implementation of component-based adaptive Web presentations. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1698–1704, New York, NY, USA, 2004. ACM Press.

Roy T. Fielding and Richard N. Taylor. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, 2002.

Robert Flenner. *Jini and Javaspaces Application Development*. Sams, 2001.

George H. Forman and John Zahorjan. The challenges of mobile computing. *IEEE Computer*, 27(4): 38–47, 1994.

Flavius Frasincar, Geert-Jan Houben, and Peter Barna. Hera presentation generator. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 952–953, New York, NY, USA, 2005. ACM Press.

Svend Frolund and Kannan Govindarajan. cl: A language for formally defining web services interactions. Technical Report HPL-2003-208, Hewlett Packard Laboratories, October 10 2003. URL `http://www.hpl.hp.com/techreports/2003/HPL-2003-208.pdf`.

Franca Garzotto, Luca Mainetti, and Paolo Paolini. Hypermedia design, analysis, and evaluation issues. *Communications of the ACM*, 38(8):74–86, 1995.

Franca Garzotto, Paolo Paolini, and Daniel Schwabe. HDM - a model-based approach to hypertext application design. *ACM Transactions on Information Systems*, 11(1):1–26, 1993.

James H. Gilmore and B. Joseph Pine. *Markets of One: Creating Customer-Unique Value through Mass Customization*. Harvard Business School Press, 2000.

Athula Ginige and San Murugesan. Guest editors' introduction: Web engineering - an introduction. *IEEE MultiMedia*, 8(1):14–18, 2001.

Aniruddha Gokhale, Bharat Kumar, and Arnaud Sahuguet. Reinventing the Wheel? CORBA vs. Web Services. In *Proceedings of the 11th International World Wide Web Conference*, 2002.

Richard Gold and Cecilia Mascolo. Use of context-awareness in mobile peer-to-peer networks. In *Proc. IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 01)*, 2001.

Asunción Gómez-Pérez, Mariano Fernández-López, and Oscar Corcho-Garcia. *Ontological Engineering*. Springer, 2004.

William G. Griswold, Robert Boyer, Steven W. Brown, and Tan Minh Truong. A component architecture for an extensible, highly integrated context-aware computing infrastructure. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 363–372. IEEE Computer Society, 2003.

John Grundy and Biao Yang. An environment for developing adaptive, multi-device user interfaces. In *CRPITS '18: Proceedings of the Fourth Australian user interface conference on User interfaces 2003*, pages 47–56. Australian Computer Society, Inc., 2003.

V. Haarslev, R. Möller, and M. Wessel. Querying the Semantic Web with Racer + nRQL. In *Proceedings of the KI-2004 International Workshop on Applications of Description Logics*. CEUR, 2004.

Frank Halasz and Mayer Schwartz. The Dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, 1994.

George T. Heineman and William T. Councill, editors. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

George T. Heineman and Helgo M. Ohlenbusch. An Evaluation of Component Adaptation Techniques. In *2nd Annual Workshop on Component-Based Software Engineering, Los Angeles, CA*, 1999.

I.T. Hernadvölgyi, G. Ucelli, O. Symonova, L. Delpero, and R. de Amicis. Shape Semantics from Shape Context. In *Proceedings of the KI-2004 International Workshop on Modelling and Retrieval of Context, Vol-114*. CEUR, 2004.

Ian Horrocks. The FaCT System. In *TABLEAUX '98: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 307–312, London, UK, 1998. Springer-Verlag.

T. Isakowitz, Edward A. Stohr, and P. Balasubramanian. RMM: a methodology for structured hypermedia design. *Communications of the ACM*, 38(8):34–44, 1995.

Jens Jahnke, Yury Bychkov, David Dahlem, and Luay Kawasme. Context-Aware Information Services for Health care. In *Proceedings of the KI-2004 International Workshop on Modelling and Retrieval of Context, Vol-114*. CEUR, 2004.

Yuhui Jin, Stefan Decker, and Gio Wiederhold. Ontowebber: Model-driven ontology-based web site management. In Isabel F. Cruz, Stefan Decker, Jérôme Euzenat, and Deborah L. McGuinness, editors, *SWWS*, pages 529–547, 2001.

Joachim Wolfgang Kaltz. Using Context-Awareness to Support User Interaction with Web Services. In *International Conference on Internet and Web Applications and Services (ICIW'06)*. IEEE, 2006.

Joachim Wolfgang Kaltz, Steffen Lohmann, Eike Lang, Tim Hussein, and Jürgen Ziegler. Ontologiebasiertes Engineering kontextadaptiver Webanwendungen. *i-com Zeitschrift für interaktive und kooperative Medien*, 3/2005:22–30, 2005a.

Joachim Wolfgang Kaltz, Steffen Lohmann, and Jürgen Ziegler. Eine komponentenorientierte Architektur für die kontextsensitive Adaption von Web-Anwendungen. In *Informatik 2005 - Informatik Live!, Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, pages 129–133, Bonn, Germany, 2005b. Köllen-Verlag.

Joachim Wolfgang Kaltz and Jürgen Ziegler. A Conceptual Model for Context-aware Web Engineering. In *Proceedings of the KI-2004 International Workshop on Modelling and Retrieval of Context, Vol-114*. CEUR, 2004.

Joachim Wolfgang Kaltz and Jürgen Ziegler. Supporting Systematic Usage of Context in Web Applications. In *The 19th International FLAIRS Conference, special track on Modeling and Applying Contexts in the Real World (MAC-06)*. AAAI, 2006.

Joachim Wolfgang Kaltz, Jürgen Ziegler, and Steffen Lohmann. Context-aware Web Engineering: Modeling and Applications. *RIA — Revue d'Intelligence Artificielle, Special Issue on Applying Context Management*, 19(3):439–458, 2005c.

Gerti Kappel, Birgit Pröll, Werner Retschitzegger, and Wieland Schwinger. Customisation for Ubiquitous Web Applications - A Comparison of Approaches. *International Journal of Web Engineering and Technology*, 1(1):79–111, 2003.

Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. RQL: a declarative query language for RDF. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 592–603, New York, NY, USA, 2002. ACM Press.

Jalal Kawash. Declarative user interfaces for handheld devices. In *WISICT '04: Proceedings of the winter international synposium on Information and communication technologies*, pages 1–6. Trinity College Dublin, 2004.

KDE Community. *Tenor*, n.d. URL `http://appeal.kde.org/wiki/Tenor`. Last visited March 16, 2006.

Markus Keidl and Alfons Kemper. Towards context-aware adaptable web services. In *Proceedings of the 13th international World Wide Web conference - Alternate Track Papers & Posters*, pages 55–65, New York, NY, USA, 2004. ACM Press.

Norbert Kiesel, Andy Schuerr, and Bernhard Westfechtel. GRAS, A Graph-Oriented (Software) Engineering Database System. *Information Systems*, 20(1):21–51, 1995.

Ross King, Niko Popitsch, and Utz Westermann. METIS: a flexible database foundation for unified media management. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 744–745. ACM Press, 2004.

Nora Koch. *Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process*. PhD thesis, Ludwig-Maximilians-University Munich, 2001.

Anders Kofod-Petersen and Agnar Aamodt. Case-based situation assessment in a mobile context-aware system. In A. Kruger and R. Malaka, editors, *Proceedings of AIMS2003, Workshop on Artificial Intelligence for Mobile Systems*, Seattle (WA), USA, 2003.

Andreas Kraus and Nora Koch. Generation of Web Applications from UML Models using an XML Publishing Framework. In *Proceedings of the 6th World Conference on Integrated Design and Process Technology (IDPT)*, volume 1, 2002.

Mik Lamming and Mike Flynn. Forget-me-not: intimate computing in support of human memory. In *Proceedings FRIEND21 Symposium on Next Generation Human Interfaces*, 1994.

Dongwon Lee and Wesley W. Chu. Comparative analysis of six XML schema languages. *SIGMOD Record*, 29(3):76–87, 2000.

Yuangui Lei, Enrico Motta, and John Domingue. Design of customized web applications with OntoWeaver. In *K-CAP '03: Proceedings of the international conference on Knowledge capture*, pages 54–61. ACM Press, 2003.

Quanzhong Li, Michelle Y. Kim, Edward So, and Steve Wood. Xvm: a bridge between xml data and its behavior. In *Proceedings of the 13th international conference on World Wide Web*, pages 155–163. ACM Press, 2004.

Henry Lieberman. Letizia: an agent that assists web browsing. In Chris S. Mellish, editor, *Proceedings IJCAI-95, 14th International Joint Conference on Artificial Intelligence*, pages 924–929. Morgan Kaufmann Publishers, 1995.

Fernanda Lima and Daniel Schwabe. Application modeling for the semantic web. In *LA-WEB '03: Proceedings of the First Conference on Latin American Web Congress*, page 93, Washington, DC, USA, 2003. IEEE Computer Society.

Hugo Liu and Pattie Maes. What would they think?: a computational model of attitudes. In *Proceedings of the 2004 International Conference on Intelligent User Interfaces*, pages 38–45, 2004.

Sushil J. Louis and Anil Shankar. Context learning can improve user interaction. In Du Zhang, Éric Grégoire, and Doug DeGroot, editors, *IRI*, pages 115–120. IEEE Systems, Man, and Cybernetics Society, 2004.

Zakaria Maamar, Soraya Kouadri, and Hamdi Yahyaoui. A Web services composition approach based on software agents and context. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1619–1623. ACM Press, 2004.

Ankur Mani, Hari Sundaram, David Birchfield, and Gang Qian. The networked home as a user-centric multimedia system. In *NRBC '04: Proceedings of the 2004 ACM workshop on Next-generation residential broadband challenges*, pages 19–30, New York, NY, USA, 2004. ACM Press.

Ioana Manolescu, Marco Brambilla, Stefano Ceri, Sara Comai, and Piero Fraternali. Model-driven design and deployment of service-enabled web applications. *ACM Transactions on Internet Technology*, 5(3):439–479, 2005.

Stefano Mazzocchi. Introducing Cocoon 2.0. Website, 2002. `http://www.xml.com/pub/a/2002/02/13/cocoon2.html`. Last visited May 20, 2006.

Brian McBride. RDF and its Vocabulary Description Language. In *Handbook on Ontologies*. Springer-Verlag, 2004.

John L. McCarthy. Notes on formalizing context. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 555–562, 1993.

Robert Meersman. Semantic Web and Ontologies: Playtime or Business at the Last Frontier in Computing ? In *Proceedings of the NSF-EU Workshop on Database and Information Systems Research for Semantic Web and Enterprises (on-line)*, pages 61–67, 2002.

Marius Mikalsen and Anders Kofod-Petersen. Representing and Reasoning about Context in a Mobile Environment. In *Proceedings of the KI-2004 International Workshop on Modelling and Retrieval of Context, Vol-114*. CEUR, 2004.

Scott Mitchell. *Designing Active Server Pages*. O'Reilly, Sebastopol, CA, USA, 2000.

Lajos Moczar and Jeremy Aston. *Cocoon Developer's Handbook*. Sams, Indianapolis, IN, USA, 2002.

Francisco Jose Monaco, Adilson Gonzaga, and Leonardo B. Guerreiro. Restraining content explosion vs. constraining content growth. In *Web Engineering, Software Engineering and Web Application Development*, pages 148–155. Springer-Verlag, 2001.

Adam Moore, Timothy J. Brailsford, and Craig D. Stewart. Personally tailored teaching in WHURLE using conditional transclusion. In *Proceedings of the twelfth ACM conference on Hypertext and Hypermedia*, pages 163–164. ACM Press, 2001.

David R. Morse, Stephen Armstrong, and Anind K. Dey. The what, who, where, when, why and how of context-awareness. In *CHI '00 extended abstracts on Human factors in computing systems*, pages 371–371. ACM Press, 2000.

Thomas J. Mowbray and William A. Ruh. *Inside Corba: Distributed Object Standards and Applications*. Addison Wesley Longman, 1997.

John Paul Mueller. *Active X From the Ground Up*. McGraw-Hill, 1997.

San Murugesan, Yogesh Deshpande, Steve Hansen, and Athula Ginige. Web Engineering: A New Discipline for Development of Web-Based Systems. In S. Murugesan and Y. Desphande, editors, *Web Engineering. Managing Diversity and Complexity of Web Application Development*, volume 2016 of *LNCS*, page 3 ff, Berlin, Heidelberg, 2001. Springer.

Jocelyne Nanard and Marc Nanard. Hypertext design environments and the hypertext design process. *Communications of the ACM*, 38(8):49–56, 1995.

Surya Nepal and Uma Srinivasan. Dave: a system for quality driven adaptive video delivery. In *Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, pages 223–230. ACM Press, 2003.

Gustav Neumann and Uwe Zdun. Distributed web application development with active web objects. In *Proceedings of The 2nd International Conference on Internet Computing*, pages 623–629, 2001.

Eric Newcomer. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley, 2002.

Hung Q. Ngo, Anjum Shehzad, Saad Liaquat, Maria Riaz, and Sungyoung Lee. Developing Context-Aware Ubiquitous Computing Systems with a Unified Middleware Framework. In *Embedded and Ubiquitous Computing: International Conference EUC 2004, Aizu-Wakamatsu City, Japan. Proceedings*, pages 672 – 681. Springer-Verlag, 2004.

Jakob Nielsen. *Hypertext and hypermedia*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

Marko Niinimaki, Miika Tuisku, and Matti Heikkurinen. Designing for flexibility: separating user interface design from program logic and data. In *Proceedings of the 1st international symposium on Information and communication technologies*, pages 475–480. Trinity College Dublin, 2003.

Daniel Oberle, Steffen Staab, Rudi Studer, and Raphael Volz. Supporting application development in the semantic web. *ACM Transactions on Internet Technology*, 5(2):328–358, 2005.

Object Management Group. *Unified Modeling Language*, n.d. URL `http://www.omg.org/technology/documents/formal/uml.htm`. Last visited October 20, 2005.

Open Mobile Alliance. *User Agent Profile*, 2003. URL `http://www.openmobilealliance.org/release_program/docs/UAProf/OMA-UAProf-V2_0` Last visited May 21, 2006.

Reinhard Oppermann, editor. *Adaptive user support: ergonomic design of manually and automatically adaptable software*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1994.

Randy J. Pagulayan, Kevin Keeker, Dennis Wixon, Ramon L. Romero, and Thomas Fuller. User-centered design in games. *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*, pages 883–906, 2003.

Daniela Petrelli, Elena Not, Massimo Zancanaro, Carlo Strapparava, and Oliviero Stock. Modelling and adapting to context. *Personal and Ubiquitous Computing*, 5(1):20–24, 2001.

Marlon Pierce, Geoffrey Fox, Choonhan Youn, Steve Mock, Kurt Mueller, and Ozgur Balsoy. Interoperable Web services for computational portals. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–12, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

Salil Pradhan. Semantic location. *Personal and Ubiquitous Computing*, 4(4):213–216, 2000.

Steve Punte. Getting Started With Cocoon 2. Website, 2002. `http://www.xml.com/pub/a/2002/07/10/cocoon2.html`. Last visited May 20, 2006.

Jean-Philippe Ramu, Yvonne Barnard, Francis Payeur, and Patrick Larroque. Contextualised operational documentation in aviation. In *Human Factors in Design*. Shaker Publishing, 2004.

Germano Resconi and Lakhmi C. Jain. *Intelligent Agents - Theory and Applications*. Springer, 2004.

Howard Rheingold. *Smart Mobs: The Next Social Revolution*. Perseus Publishing, 2002.

John Riedl and Joseph Konstan. *Word of Mouse: The Marketing Power of Collaborative Filtering*. Warner Business Books, 2002.

James Robertson. Content reuse in practice. Website, 2004. `http://www.steptwo.com.au/papers/kmc_contentreuse/pdf/KMC_ContentReuse.pdf`. Last visited March 13, 2006.

Cristiano Rocha, Daniel Schwabe, and Marcus Poggi Aragao. A hybrid approach for searching in the semantic web. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 374–383, New York, NY, USA, 2004. ACM Press.

Matthew J. Rutherford and Alexander L. Wolf. A case for test-code generation in model-driven systems. In *GPCE '03: Proceedings of the second international conference on Generative programming and component engineering*, pages 377–396, New York, NY, USA, 2003. Springer-Verlag New York, Inc.

Michael Samulowitz, Florian Michahelles, and Claudia Linnhoff-Popien. Adaptive interaction for enabling pervasive services. In *Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access*, pages 20–26. ACM Press, 2001.

J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1-2):115–153, 2001.

Arno Scharl. *Evolutionary Web Development*. Springer-Verlag, Secaucus, NJ, USA, 2000.

Herbert Schildt. *Java: The Complete Reference, J2SE 5 Edition*. McGraw-Hill Osborne Media, 2004.

Bill Schilit and M. Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5):22–32, 1994.

Bill N. Schilit, Anthony LaMarca, Gaetano Borriello, William G. Griswold, David McDonald, Edward Lazowska, Anand Balachandran, Jason Hong, and Vaughn Iverson. Challenge: ubiquitous location-aware computing and the "place lab" initiative. In *WMASH '03: Proceedings of the 1st ACM international workshop on Wireless mobile applications and services on WLAN hotspots*, pages 29–35, New York, NY, USA, 2003. ACM Press.

Albrecht Schmidt. Implicit Human Computer Interaction Through Context. In *Personal Technologies Volume 4(2 and 3)*, pages 192 – 199, 2000.

Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to context than location. *Computers and Graphics*, 23(6):893–901, 1999.

Albrecht Schmidt and Kristof Van Laerhoven. How to build smart appliances. *IEEE Personal Communications*, pages 66–71, 2001.

Barbara Schmidt-Belz, Heimo Laamanen, Stefan Poslad, and Alexander Zipf. Location-based mobile tourist services - first user experiences. In A. J. Frew, M. Hitz, and P. O'Connor, editors, *Proceedings of the 10th International Conference on Information Technologies in Tourism (ENTER 2003)*, Helsinki, Finland, 2003. Springer-Verlag.

Jean-Guy Schneider and Jun Han. Components - the Past, the Present, and the Future. In Wolfgang Weck Clemens Szyperski and Jan Bosch, editors, *Proceedings of Ninth International Workshop on Component-Oriented Programming*, 2004.

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

Ahmed Seffah and Eduard Metzker. The obstacles and myths of usability and software engineering. *Communications of the ACM*, 47(12):71–76, 2004.

Michael Semrau and Achim Kraiss. Mobile commerce for financial services—killer applications or dead end? *SIGGROUP Bulletin*, 22(1):22–25, 2001.

Humphrey Sheil. Distributed scientific computing in java: observations and recommendations. In *PPPJ '03: Proceedings of the 2nd international conference on Principles and practice of programming in Java*, pages 219–222, New York, NY, USA, 2003. Computer Science Press, Inc.

B.I.J. Siljee, I.E. Bosloper, and J.A.G. Nijhuis. A Classification Framework for Storage and Retrieval of Context. In *Proceedings of the KI-2004 International Workshop on Modelling and Retrieval of Context, Vol-114*. CEUR, 2004.

Aaron Skonnard and Martin Gudgin. *Essential XML Quick Reference: A Programmer's Reference to XML, XPath, XSLT, XML Schema, SOAP, and More*. Addison-Wesley, 2001.

Peter Spyns, Robert Meersman, and Mustafa Jarrar. Data modelling versus ontology engineering. *SIGMOD Record*, 31(4):12–17, 2002.

Christopher Staff. Hypercontextr: A model for adaptive hypertext. In A. Jameson, C. Paris, and C. Tasso, editors, *Proceedings of the Sixth International Conference on User Modeling*, pages 33–39. Springer, Berlin, 1997.

Christopher D Staff. The hypercontext framework for adaptive hypertext. In *HYPERTEXT '02: Proceedings of the thirteenth ACM conference on Hypertext and hypermedia*, pages 11–20. ACM Press, 2002.

Stanford Medical Informatics. *The Protégé Ontology Editor and Knowledge Acquisition System*, n.d. URL `http://protege.stanford.edu/`. Last visited March 8, 2006.

Martin Staudt, Jorg-Uwe Kietz, and Ulrich Reimer. A data mining support environment and its application on insurance data. In *Knowledge Discovery and Data Mining*, pages 105–111, 1998.

Thomas Strang and Claudia Linnhoff-Popien. A Context Modeling Survey. In *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*, 2004.

Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. Applications of a Context Ontology Language. In Dinko Begusic and Nikola Rozic, editors, *Proceedings of International Conference on Software, Telecommunications and Computer Networks (SoftCom2003)*, pages 14–18. University of Split, Croatia, 2003a.

Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. CoOL: A Context Ontology Language to enable Contextual Interoperability. In Jean-Bernard Stefani, Isabelle Dameure, and Daniel Hagimont, editors, *Proceedings of 4th International Conference on Distributed Applications and Interoperable Systems (DAIS2003)*, volume 2893 of *Lecture Notes in Computer Science (LNCS)*, pages 236–247, Paris, France, 2003b. Springer.

Jun-Zhao Sun and Jaakko Sauvola. Towards a conceptual model for context-aware adaptive services. In *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 90 – 94, 2003.

Koichi Terai, Noriaki Izumi, and Takahira Yamaguchi. Coordinating Web Services based on business models. In *Proceedings of the 5th international conference on Electronic commerce*, pages 473–478. ACM Press, 2003.

Scott M. Thayer and Peter Steenkiste. An Architecture for the Integration of Physical and Informational Spaces. *Personal and Ubiquitous Computing*, 7(2):82–90, 2003.

The Apache Software Foundation. *The Apache Cocoon Project*, n.d.a. URL `http://cocoon.apache.org`. Last visited September 09, 2005.

The Apache Software Foundation. *Apache Excalibur*, n.d.b. URL `http://excalibur.apache.org`. Last visited September 09, 2005.

The Apache Software Foundation. *Struts*, n.d.c. URL `http://struts.apache.org`. Last visited September 09, 2005.

The Apache Software Foundation. *WebServices - Axis*, n.d.d. URL `http://ws.apache.org/axis/`. Last visited September 09, 2005.

Jerome Thomere, Ken Barker, Vinay Chaudhri, Peter Clark, Michael Eriksen, Sunil Mishra, Bruce Porter, and Andres Rodriguez. A web-based ontology browsing and editing system. In *Eighteenth national conference on Artificial intelligence*, pages 927–934. American Association for Artificial Intelligence, 2002.

Michael E. Tipping. Sparse bayesian learning and the relevance vector machine. *Machine Learning Research*, 1:211–244, 2001.

Wolfgang Trumler, Faruk Bagci, Jan Petzold, and Theo Ungerer. Smart Doorplates - Toward an Autonomic Computing System. *Personal and Ubiquitous Computing*, 7(3-4):221–226, 2003.

Eric van der Vlist. *XML Schema*. O'Reilly, Sebastopol, CA, USA, 2002.

Eric van der Vlist. *RELAX NG*. O'Reilly, Sebastopol, CA, USA, 2003.

Jacco van Ossenbruggen, Joost Geurts, Frank Cornelissen, Lynda Hardman, and Lloyd Rutledge. Towards second and third generation web-based multimedia. In *Proceedings of the tenth international conference on World Wide Web*, pages 479–488. ACM Press, 2001.

Chris Vandervelpen and Karin Coninx. Towards model-based design support for distributed user interfaces. In *NordiCHI '04: Proceedings of the third Nordic conference on Human-computer interaction*, pages 61–70, New York, NY, USA, 2004. ACM Press.

Steve Vandivier and Kelly Cox. *Oracle9i Application Server Portal Handbook*. Osborne/McGraw-Hill, 2001.

Giannis Varelas, Epimenidis Voutsakis, Paraskevi Raftopoulou, Euripides G.M. Petrakis, and Evangelos E. Milios. Semantic similarity methods in wordnet and their application to information retrieval on the web. In *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 10–16, New York, NY, USA, 2005. ACM Press.

Richard Vdovjak, Flavius Frasincar, Geert-Jan Houben, and Peter Barna. Engineering semantic web information systems in hera. *Journal of Web Engineering*, 2(1-2):3–26, 2003.

Carl Vieregger. Product Roundup: A Peck of Java Portlets. *Software Development*, (3):27–29, 2003.

W3C. SPARQL Query Language for RDF. Working Draft, http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/, W3C, 2005.

W3C Device Independence Working Group. *Device Independence Activity*, n.d. URL `http://www.w3.org/2001/di/`. Last visited April 4, 2006.

Ethan Watrall. *Dreamweaver MX: Design and Technique*. Sybex Inc, 2002.

Christian Wege. Portal server technology. *IEEE Internet Computing*, 6(3):73–77, 2002.

Florian Wegscheider, Thomas Dangl, Michael Jank, and Rainer Simon. A multimodal interaction manager for device independent mobile applications. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 272–273. ACM Press, 2004.

Graham Wilcock. Integrating Natural Language Generation with XML Web Technology. In *Proceedings of the Demo Sessions of EACL-2003, Budapest*, pages 247 – 250, 2003.

Terry Winograd. Architectures for Context. *Human-Computer Interaction*, 16:401–419, 2001.

WISE. *Web Information and Service Engineering, Bundesministerium für Bildung und Forschung (BMBF) funded project (Nr. 01ISC30F)*, n.d. URL `http://www.wise-projekt.de`. Last visited May 20, 2006.

Michael Wissen. Eine komponentenorientierte Vorgehensweise zur Modellierung von Websites. *i-com*, 4(1):13–21, 2004a.

Michael Wissen. *Eine Vorgehensweise zur systematischen Modellierung web-basierter Anwendungen*. Dissertation, Institut für Arbeitswissenschaft und Technologiemanagement der Universität Stuttgart, Stuttgart, Germany, 2004b. URL `http://elib.uni-stuttgart.de/opus/volltexte/2005/2272/`.

Michael Wissen and Jürgen Ziegler. A Methodology for the Component-Based Development of Web Applications. In *Proceedings of 10th Int. Conf. on Human - Computer Interaction (HCI International 2003), Vol. 1, Crete, Greece*, 2003.

Tomoko Yonezawa, Brian Clarkson, Michiaki Yasumura, and Kenji Mase. Context-aware sensor-doll as a music expression device. In *CHI '01 extended abstracts on Human factors in computing systems*, pages 307–308. ACM Press, 2001.

M. R. Zakaria, A. Moore, C. D. Stewart, and T. J. Brailsford. "Pluggable" user models for adaptive hypermedia in education. In *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, pages 170–171. ACM Press, 2003.

Jia Zhang, Jen-Yao Chung, and Carl K. Chang. Towards increasing web application productivity. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 1677–1681, New York, NY, USA, 2004. ACM Press.

Carsten Ziegeler and Matthew Langham. *Cocoon: Building XML Applications*. Sams, Indianapolis, IN, USA, 2002.

Jürgen Ziegler, Christoph Kunz, and Veit Botsch. Matrix browser: visualizing and exploring large networked information spaces. In *CHI '02 extended abstracts on Human factors in computing systems*, pages 602–603. ACM Press, 2002.

Jürgen Ziegler, Steffen Lohmann, and Joachim Wolfgang Kaltz. Kontextmodellierung für adaptive webbasierte Systeme. In C. Stary, editor, *Mensch & Computer 2005: Kunst und Wissenschaft - Grenzüberschreitungen der interaktiven ART*, pages 181–189, München, Germany, 2005. Oldenbourg.

Olaf Zimmermann, Vadim Doubrovski, Jonas Grundler, and Kerard Hogg. Service-oriented architecture and business process choreography in an order management scenario: rationale, concepts, lessons learned. In *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 301–312, New York, NY, USA, 2005. ACM Press.

Zope Community. *Zope.org*, n.d. URL `http://www.zope.org`. Last visited March 16, 2006.