User Modeling and User Profiling in Adaptive E-learning Systems

An approach for a service-based personalization solution for the research project AdeLE (Adaptive e-Learning with Eye-Tracking)

Master's Thesis
at
Graz University of Technology

submitted by

Christoph Fröschl

Institute for Information Systems and Computer Media (IICM)
Faculty of Computer Science
Graz University of Technology
A-8010 Graz, Austria

November 2005

This thesis is written in English language.

Assessor:	o.Univ-Prof. Dr. Dr.h.c.mult. Hermann Maurer
Supervisor:	DI Victor Manuel Garcia-Barrios
Co-Supervisor:	DI Dr. techn. Christian Gütl

Benutzermodellierung und Benutzerprofile in Adaptiven E-learning Systemen

Ein Ansatz für eine service-basierte Personalierung im Zuge des Forschungsprojekts AdeLE (Adaptive e-Learning with Eye-Tracking)

Magister Arbeit
an der
Technischen Universität Graz

vorgelegt von

Christoph Fröschl

Institut für Informationssysteme und Computer Medien (IICM)
Fakultät für Information
Technische Universität Graz
A-8010 Graz, Österreich

November 2005

Diese Arbeit ist in englischer Sprache verfasst.

Begutachter:	o.Univ-Prof. Dr. Dr.h.c.mult. Hermann Maurer
Betreuer:	DI Victor Manuel Garcia-Barrios
Mitbetreuer:	DI Dr. techn. Christian Gütl

Abstract

It is essential for user-adaptive systems to have information about the user. Without any information about the user the adaptive system is not able to adapt itself to the user's characteristics and preferences. The required information is stored and managed in form of user models. Thus, a user model represents the system's beliefs about the user. The construction and the content of user models is the main issue of this work. Further, a possible solution for a user modeling system will be introduced and described.

The theoretical part of this thesis addresses all necessary aspects of a user model. This enables the investigation and examination of available and well-founded solution approaches. Important issues in this context are role and content of user models as well as user modeling methods, which are utilized for the construction and administration of a user model. Further, existing standards in the field of user modeling and their applicability are described. Finally, existing user modeling systems are examined under the aspects of previous findings.

The proposed solution approach of the practical part of this thesis is based on a service-oriented architecture. Important issues of a user modeling system are, among others, modularity and flexibility. The service-oriented architecture enables the implementation of these features. Further, issues like privacy and security as well as organization of user data are seen as indispensable requirements concerning integrity and exploitability of user data within user modeling systems, and are therefore included in the proposed solution approach as well.

Several service-oriented frameworks are introduced, whereby *Openwings* was utilized within the solution approach. Regarding the design of service-oriented systems, an answer to the questions "how big" a service should be and "how much" functionality a service should offer is necessary. A first approach to answer these questions is done by creating two different implementations of the proposed solution. These implementations represent two extrema concerning size and functionality of services. An evaluation followed by a comparison of the evaluation results show advantages and drawbacks of these implementations.

Kurzfassung

Für benutzeradaptive Systeme ist es essentiell relevante Informationen über die Benutzer zu haben. Ohne Informationen über den Benutzer ist es unmöglich für ein adaptives System sich an die Eigenschaften und Vorlieben des jeweiligen Benutzers anzupassen. Diese benötigten Informationen werden in Form von Benutzermodellen gespeichert und verwaltet. Ein Benutzermodel stellt die systeminterne Abbildung des Benutzers dar. Der Aufbau und der Inhalt von Benutzermodellen stellt den Kernpunkt dieser Arbeit. Weiters wird ein Lösungsansatz für ein Benutzermodellierungssystem erarbeitet und vorgestellt.

Im theoretischen Teil der vorliegenden Arbeit werden alle nötigen Aspekte eines Benutzermodels untersucht, um nach vorhandenen bzw. etablierten Lösungsansätzen zu suchen. Wichtige Punkte stellen dabei die Aufgabe und der Inhalt eines Benutzermodels bzw. die Methoden, die bei der Erstellung und Verwaltung eines Benutzermodels verwendet werden, dar. Weiters werden bestehende Standards im Bereich Benutzermodellierung und deren Anwendbarkeit untersucht, um schließlich bereits bestehende Benuzermodellierungssysteme aus Sicht dieser Aspekte beleuchten zu können.

Der vorgestellte Lösungsansatz im praktischen Teil dieser Arbeit basiert auf einer service-orientierten Architektur, mit der es möglich ist die für wichtig befundenen Eigenschaften eines Benutzermodellierungssystems umzusetzen. Wichtige technische Eigenschaften sind unter anderem Modularität und Flexibilität. Weitere Anforderungen wie "Privacy/Security" und Organisation von Benutzerdaten sind im Bereich Benutzermodellierung hinsichtlich Integrität und Verwendbarkeit der Benutzerinformation von enormer Wichtigkeit, und werden ebenfalls im vorgestellten Lösungsansatz eingearbeitet.

Als Grundlage für den Lösungsansatz werden mehrere service-orientierte Rahmenwerke vorgestellt, wobei *Openwings* für den Lösungsansatz herangezogen wurde. Bezüglich dem Design von service-orientierten Systemen besteht der Bedarf nach einer Antwort auf die Fragen "wie groß" ein Service sein bzw. "wie viel" Funktionalität ein Service zur Verfügung stellen soll. Eine erste Näherung an diese Fragen wird mittels des Entwurfs zweier Varianten des Lösungsansatzes durchgeführt. Die beiden Varianten stellen zwei Extrema hinsichtlich Größe und Funktionsumfang von Services dar. Eine Evaluierung und ein darauf folgender Vergleich der Ergebnisse gibt Aufschlüsse über die Vor- und Nachteile beider Varianten.

Ich versichere hiermit, diese Arbeit selbständig verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

I hereby certify that the work presented in this thesis is my own and that work performed by others is appropriately cited.

Signature of the author: _____

Contents

1 Introduction **5**
- 1.1 Initial Situation and Motivation . . . 5
- 1.2 Structure of this Work . . . 7

2 Basic Principles **9**
- 2.1 Introduction . . . 9
- 2.2 User Profiling and User Modeling . . . 9
  - 2.2.1 The Difference between User Profiling and User Modeling . . . 9
  - 2.2.2 Necessity of User Profiling and User Modeling . . . 10
- 2.3 Terminology . . . 11
  - 2.3.1 Adaptive, Adaptable and Personalization . . . 11
  - 2.3.2 E-learning . . . 12
  - 2.3.3 Adaptive Systems . . . 12
- 2.4 Adaptive E-learning Systems . . . 13
  - 2.4.1 Theoretical Approaches . . . 14
  - 2.4.2 Types of Systems . . . 16
- 2.5 Summary . . . 18

3 User Modeling in Adaptive Systems **20**
- 3.1 Introduction . . . 20
- 3.2 The Role of User Models in Adaptive Systems . . . 20
- 3.3 User Models in Adaptive E-learning Systems . . . 22
  - 3.3.1 User Models in Macro-adaptive Systems . . . 22
  - 3.3.2 User Models in ITS . . . 23
  - 3.3.3 User Models in AEHS . . . 25
- 3.4 Learner Modeling . . . 27
  - 3.4.1 Content of a Learner Model . . . 27
  - 3.4.2 Components of a Learner Model . . . 31
- 3.5 Modeling Techniques . . . 32
  - 3.5.1 Methods to Construct Learner Models . . . 32
  - 3.5.2 Initialization of Learner Models . . . 36
  - 3.5.3 Update of Learner Models . . . 37
- 3.6 Summary . . . 39

4 Standards for User Modeling and Profiling **41**
- 4.1 Introduction . . . 41
- 4.2 Basic Standards . . . 41
  - 4.2.1 vCard . . . 41
  - 4.2.2 eduPerson . . . 42
  - 4.2.3 Universal Learning Format (ULF) . . . 42
- 4.3 GESTALT . . . 42
  - 4.3.1 Architecture . . . 43

4.3.2 Data Model . . . 45
4.4 Public and Private Information - PAPI Learner . . . 46
4.4.1 Common Features, Information Types and Bindings . . . 46
4.4.2 PAPI Learner Information Groups . . . 48
4.4.3 Public and Private Information . . . 49
4.4.4 Summary . . . 50
4.5 IMS Learner Information Package . . . 50
4.5.1 The Structure of IMS LIP . . . 51
4.5.2 XML Schema . . . 53
4.5.3 Data Protection . . . 56
4.5.4 Implementations . . . 57
4.5.5 Summary . . . 57
4.6 Summary and Conclusion . . . 58

5 State of the Art of User Modeling Systems 59
5.1 Introduction . . . 59
5.2 Early User Modeling Systems . . . 59
5.2.1 GUMS . . . 60
5.3 User Modeling Shell Systems . . . 62
5.3.1 UMT . . . 62
5.3.2 PROTUM . . . 63
5.3.3 um . . . 63
5.3.4 BGP-MS . . . 63
5.4 User Modeling Servers . . . 66
5.4.1 Doppelgänger . . . 66
5.4.2 Personis . . . 68
5.4.3 LDAP-based User Modeling Server . . . 70
5.4.4 Web Service and Agent-based User Modeling System . . . 72
5.5 Summary and Conclusion . . . 74

6 Basic Reflections for the Solution Approach 77
6.1 Introduction . . . 77
6.2 Service-oriented Architecture . . . 77
6.2.1 Web Services . . . 79
6.2.2 Service-oriented Frameworks . . . 81
6.2.3 Service-oriented Software Design . . . 84
6.3 Privacy and Security in User Modeling . . . 85
6.3.1 Privacy . . . 85
6.3.2 Security . . . 87
6.3.3 Privacy Technologies . . . 87
6.4 Levels of Profiling the Learner . . . 88
6.4.1 Organizing the Learner Profile . . . 88
6.4.2 Partitioning the Learner Model . . . 89
6.5 Summary and Conclusion . . . 90

7 The Openwings Framework 92
7.1 Introduction . . . 92
7.2 Overview of Openwings . . . 92
7.2.1 Openwings Core Services . . . 93
7.2.2 Contexts in Openwings . . . 97
7.2.3 Interfaces . . . 97
7.3 Components in Openwings . . . 99
7.3.1 Parts of Openwings Components . . . 99
7.3.2 The different Types of Components . . . 100
7.3.3 Lifecycle of Components . . . 101
7.3.4 Relations between Components . . . 102
7.4 Software Development with Openwings . . . 103
7.4.1 Creation and Intializiation of Service Objects . . . 103
7.4.2 Usage of Service Objects . . . 104
7.4.3 Policy Concept . . . 105
7.4.4 Connectors . . . 107
7.4.5 Installable Component Descriptors . . . 107
7.4.6 Summarization . . . 110
7.5 Security in Openwings . . . 110
7.5.1 Code Security . . . 110
7.5.2 Transport Security . . . 111
7.5.3 Service Security . . . 111
7.6 Conclusion . . . 112

8 Design and Implementation of the Modeling System 114
8.1 Introduction . . . 114
8.2 Software Requirements of the Modeling System . . . 114
8.2.1 Functional Requirements . . . 115
8.2.2 Non-Functional Requirements . . . 116
8.3 Architectural Design and Use Cases . . . 117
8.3.1 Use Cases for the Modeling System . . . 119
8.4 Macro-approach . . . 120
8.4.1 Functional Components . . . 120
8.4.2 Implementation of selected Aspects . . . 124
8.5 Micro-approach . . . 126
8.5.1 Functional Components . . . 126
8.5.2 Implementation of selected Aspects . . . 130
8.6 Evaluation of the Approaches . . . 132
8.6.1 User Scenarios . . . 132
8.6.2 Evaluation Criteria . . . 134
8.6.3 Evaluation Setup . . . 136
8.6.4 Evaluation Results . . . 137
8.7 Summary . . . 142

9 Summary and Outlook 143

A Additional UML Diagrams **146**

B Screen shots **154**

List of Figures **156**

List of Tables **158**

Listings **159**

Bibliography **160**

1. Introduction

1.1 Initial Situation and Motivation

The research project Adaptive e-Learning with Eye-Tracking (AdeLE[1]) is a joint scientific work of the FH Joanneum/Department of Information Design[2] and the Institute for Information Systems and Computer Media (IICM[3]) of the Graz University of Technology. AdeLE focuses on the enhancement of e-learning technologies for the purpose of improving personalization of instruction and learning content. Furthermore, an additional objective is to combine the advantages of utilizing a real-time eye-tracking system and a dynamic background library for the computation of personalization procedures and for the enhancement of knowledge transfer processes.

The eye-tracking system is used to record the behavior of the user in form of information about his gaze-movements. This information is stored in the user model and used to derive the actually consumed learning units. Figure 1.1 shows the architecture of the AdeLE system.

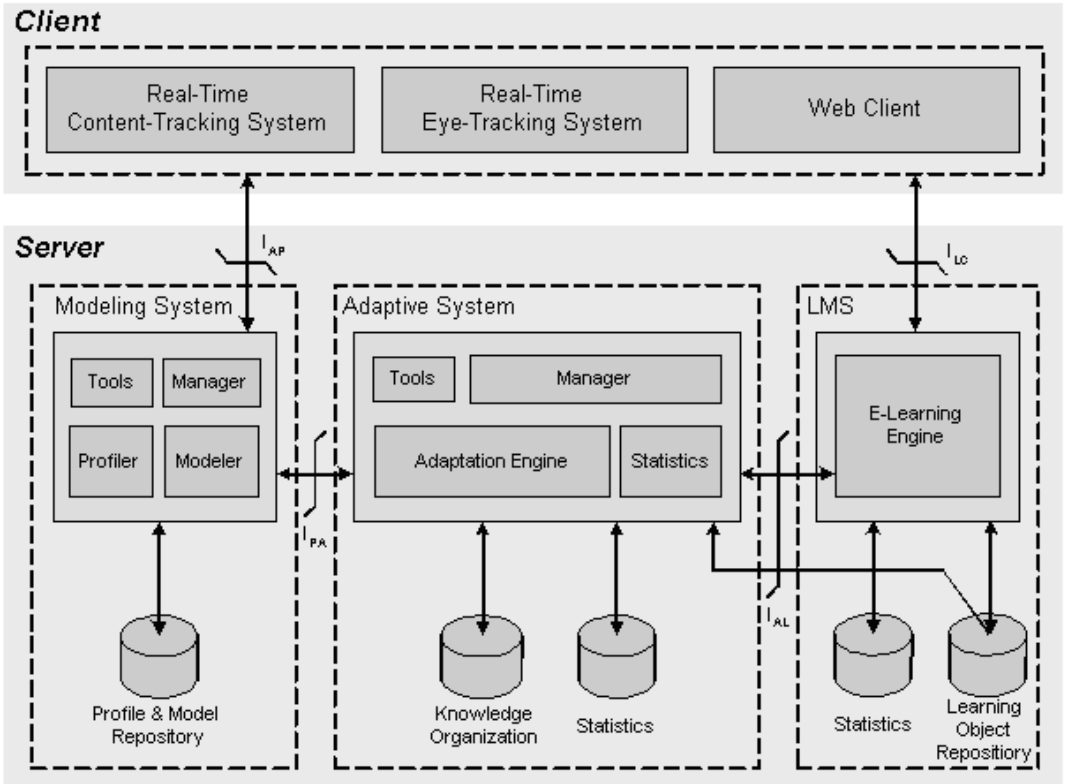

Figure 1.1: Architecture of the AdeLE system [Gütl and Garcia-Barrios 2005]

[1] http://adele.fh-joanneum.at/
[2] http://informations-design.fh-joanneum.at/
[3] http://www.iicm.edu

The AdeLE architecture is built up of strong separated client-side and server-side systems. The client-side consists of three systems, namely a real-time Content-Tracking System (CS), a real-time Eye-Tracking System (ES) and a Web Client (WC). The Web Client (WC) provides content, control and navigation elements. The real-time Eye-Tracking System (ES) monitors eye movements and delivers this behavior data to the Modeling System (MS) on the server-side. The information from the ES is linked with the CS to produce information about learning assets. The server-side of the AdeLE architecture consists of the Modeling System (MS), the Adaptive System (AS) and the LMS. The LMS compiles the learning content from the learning object repository and provides the control and navigation interface. The AS is responsible for the adaptation of the learning content, the navigation and the visualization. This adaptation is based on information from the MS. The main MS tasks are to handle user information received from the ET, CT and the AS and to generate assumptions about the user. Assumptions about the user are based on the stored data in the MS. [Gütl and Garcia-Barrios 2005]

The aim of this thesis is to design and implement a prototype of the *Modeling System* based on the main requirements for the architecture of the AdeLE System, as illustrated in Figure 1.1 (see [Gütl and Garcia-Barrios 2005] for details).

Although the concept of user modeling has a long history, the term user model itself was introduced in the 1980es. Previously approaches to adaptive systems applied user models in form of integrated units, where the user model itself was not self-contained but distributed within the whole adaptive system. During the last decade user-centered applications (e.g. recommender systems) have become very popular and made the application of sophisticated user models reasonable for the new "e-business lifestyle".

The challenge regarding the creation of user modeling systems is their multi-purpose utilization. The goal of a multi-purpose user modeling system is to satisfy the needs for several different application areas. In most cases, the user modeling systems are designed and implemented for a specific application field (e.g. e-learning) but the proposed user modeling system in this thesis should be versatile.

Further, a user modeling system must provide techniques to derive the recorded information and conclude a model for a particular user. This implies a well-structured arrangement of the user data and of the inference processes. Additionally, it is necessary to determine the required user data for the inference processes. It does not make sense to record information about a user which has no use, although nearly every piece of information helps to describe the model of the user. Storage, extraction and throughput issues limit the amount of recordable user information and makes a constriction of the user modeling system necessary. In this case, standards may help to structure the user data and determine the essential user information.

There exist numerous approaches for user modeling systems with different focuses and purposes. An examination of these systems should help to find the appropriate design for the proposed user modeling system. Special issues of the proposed solution are the service-oriented architecture, privacy and security matters as well as the structuring of the gathered user information.

Another relevant aspect for user modeling systems is their expandability. The

expandability is necessary to provide the optimal user modeling system for each application. Reconfiguring and adding modeling components to the system are essential abilities to provide an expandable user modeler. The consequences of the applied service-based architecture regarding expandability should be addressed.

The described aspects in this section and further specific issues as well as their influence on the technical solution should be examined.

1.2 Structure of this Work

Basically, this thesis is divided into a theoretical part, including the Chapters 2 - 5 and a practical part in Chapters 6 - 8. The theoretical part constitutes the analytical fundament to examine and describe required knowledge and aspects in the field of adaptive e-learning, while the practical work is based on this knowledge and ends with the proposed solution in Chapter 8.

Chapter 2 *Basic Principles* introduces basic principles like the terminology and the idea behind adaptive e-learning systems. The definition of the terminology for this work is required, since there are several different meanings used in literature for the same word. By listing different approaches and systems types, Chapter 2 gives an introduction into the fundamentals of adaptive e-learning systems.

Using the finding of Chapter 2 that personalized systems require a representation about the user in form of a user model, the Chapter 3 *User Modeling in Adaptive Systems* describes the utilization of user models in adaptive systems. Concerning adaptive e-learning systems, the users are students or learners. Therefore, the content and the structure of learner models are described. In order to apply a user model it is necessary to construct, initialize and keep the user model up-to-date. For each of these steps exist modeling techniques which are depicted at the end of Chapter 2.

There is no real definition of which content should be stored within a user model, therefore Chapter 4 *Standards for User Modeling and Profiling* describes present standards in this field focusing on structure, content, privacy precautions and implementations.

The last chapter of the theoretical part - Chapter 5 *State of the Art* - covers the state-of-the-art concerning user modeling systems. Basically, there exist numerous user modeling systems which can be grouped into shell systems and modeling servers. These systems are described by emphasizing their application focus and their architecture.

Additional issues, which are not covered by the theoretical part of this work, but are important for the proposed solution, are described in Chapter 6 *Basic Reflections for the Solution Approach*. Technical subjects, like service-oriented architecture and privacy/security methods are depicted as well. These reflections are needed for the solution approach of Chapter 8.

The proposed solution approach of Chapter 8 is based on a service-oriented framework called Openwings, which is described in Chapter 7 *The Openwings Framework*. Especially interesting aspects like security and communication are examined

concerning the utilization of Openwings for the development of a user modeling system.

Chapter 8 *Design and Implementation of the Modeling System* describes the software design and the implementation process of the solution approach. Two different approaches are depicted based on different design assumptions, namely the micro- and the macro-approach. The micro-approach follows the guideline of wrapping each component into a single services, while the macro-approach combines several components into one service. An evaluation and comparison of these two solution approaches should allow to discover the advantages and drawbacks of both approaches regarding important aspects in the field of user modeling systems.

2. Basic Principles

2.1 Introduction

Adaptive e-learning systems often employ models of the user. A user model is an internal representation of the user's properties. Before a user model can be used it has to be constructed. This process requires much efforts to gather the required information and finally generate a model of the user. User modeling is depicted in detail in Chapter 3. The aim of this chapter is to introduces basic principles like the terminology and the idea behind adaptive e-learning systems.

The difference between the terms user profiling and user modeling as well as the need for user models within adaptive systems is depicted in Section 2.2. Subsequently, the meaning of relevant terms is explained in Section 2.3 as they are used within this thesis. Finally, an overview over theoretical approaches and different types of adaptive e-learning systems is given in Section 2.4.

2.2 User Profiling and User Modeling

The terms user profiling and user modeling are often used as synonyms or only one term is used by meaning of both. To be able to clearly differentiate between a user profile and a user model the following sub-section is dedicated to describe this terminology. Reasons for constructing and using a profile or a model of the user and their applications are described in Section 2.2.2.

2.2.1 The Difference between User Profiling and User Modeling

The difference between user profiling and user modeling lies in the different level of sophistication. [Koch 2000] describes a user profile as a simple user model.

A user profile is a collection of personal information. The information is stored without adding further description or interpreting this information. It is comparable to a getting-setting mechanism of classes in object-oriented programming, where different parameters are set or retrieved. User profiles represent cognitive skills, intellectual abilities, intentions, learning styles, preferences and interactions with the system. These properties are stored after assigning them values. These values may be final or change over time.

Depending on the content and the amount of information about the user, which is stored in the user profile, a user can be modeled. Thus, the user profile is used to retrieve the needed information to build up a model of the user. [Koch 2000] describes a user model as the representation of the system's beliefs about the user. The "real world" user is perceived by the system through the human computer interface (see Figure 2.1).

The model of the user is based on this information and is therefore only a small

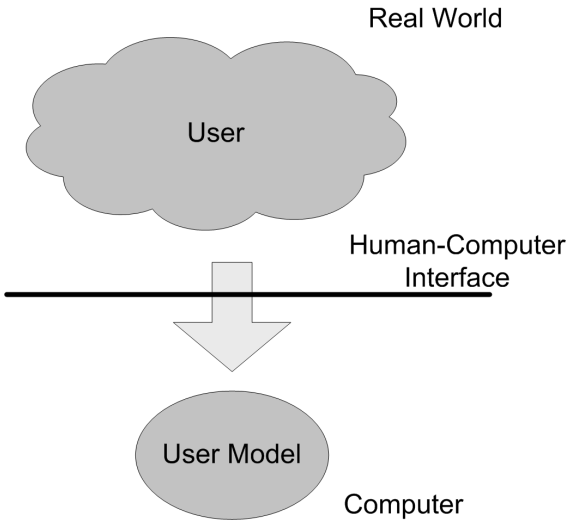

Figure 2.1: User and User Model [Kay 2000b]

part of the real user. Nevertheless, the user model must represent the needed characteristics of the user regarding the context of the application.

2.2.2 Necessity of User Profiling and User Modeling

The behavior of an adaptive system varies according to the data from the user model and the user profile. Without knowing anything about the user, a system would perform in exactly the same way for all users. [Koch 2000] describes the application of user models as follows:

> *"Users are different: they have different background, different knowledge about a subject, different preferences, goals and interests. To individualise, personalise or customise actions a user model is needed that allows for selection of individualised responses to the user."*

Therefore, everywhere where an individualized response of the system is expected, a user model should be applied.

Different types of applications can benefit from user models. User models are often components of adaptive e-learning systems and are strongly connected to the instructional part of such systems. Other applications of user modeling are for example, search engines, recommender systems or help systems.

Further, not only the attributes of a user (e.g. domain knowledge, preferences, goals, etc.) but also limitations (e.g. disabilities like color blindness etc.) to the user's perception must be considered within a user model. If these limitations have to be violated it is important to know the least disturbing options.

In the field of adaptive e-learning there are a lot of different terms used while the meaning of these terms is not clearly defined. The following section aims to clarify this problem and defines the terminology used within this thesis.

2.3 Terminology

In this section the terminology used in this diploma thesis is introduced. Terms like *adaptive*, *personalization*, *e-learning* and *adaptive systems* are described in their meaning. This is necessary since a lot of different interpretations for theses terms are used in literature.

2.3.1 Adaptive, Adaptable and Personalization

According to the [Oxford Advanced Learner's Dictionary 2005], the term *adaptive* is defined as:

> *"**adaptive** adj.: (technical) concerned with changing; able to change when necessary in order to deal with different situations"*

So *adaptive* is a property, which defines the ability to change, to suit different conditions. In other words, something is adaptive, if it is able to change itself or something else, to fit to several circumstances.

In the context of e-learning systems we can distinguish, according to [Oppermann *et al.* 1997], between two different types of adjustment. First, we have *adaptable* systems, which refer to the property of changing system parameters. The user is able to change the behavior of the system. In other words, the user is able to modify the system in specified ways to fit the users needs.

Second, the term *adaptive* means the automatic tailoring of the system to the user. The needs of the user are assumed by the system itself. The user is not asked to change system parameters to its own needs, rather the necessities of the user are supposed by the system. The system changes its behavior according to this necessities. [Oppermann *et al.* 1997]

[Weibelzahl 2003] demands also another feature. Adaptive systems obtain information about the user from observing the user.

In e-learning systems, both terms *adaptable* and *adaptive* can be used following the above stated definitions.

Personalization can be provided by tailoring the content or the visualization of the system to the user's preferences. According to [Weibelzahl 2003], the term *personalization* represents the terms adaptivity and adaptability as synonyms. Thus, both types of systems, adaptive and adaptable system, can be summarized as personalized systems.

According to [Kim 2002], there are at least two distinct origins of the term *personalization*. Firstly, dealing with the huge amount of information available today, it is necessary to gather and deliver only the information that is relevant to an individual or a group of individuals in the format and layout specified and in time intervals specified by the user. The second application of the term *personalization* is the concept of one-to-one marketing in which a business does marketing tailored to a group of individual customers. This kind of *personalization* is motivated by a rise of the revenue of the business. The customer benefits through receiving useful

and timely recommendations for purchasing goods or services in the most favorable terms.

Considering the topic of e-learning it is also necessary to deliver relevant information for the learner. Here, relevant information is the learning content, which is taught during instruction decreased by the already existing domain knowledge of the individual learner.

2.3.2 E-learning

The term *e-learning* originates from electronic learning and is often used as another term for web-based learning, online learning or distance learning. However, there are differences in the meaning of these terms. Thus, they cannot be used as interchangeable synonyms.

There are still discussion about the definition of the term *e-learning*. [Dietinger 2003] states that *e-learning* represents just one part of the learning process. It has to be completed by e-teaching while the overall process is called e-education. However, the common meaning of *e-learning* includes the overall process as well and within this thesis only the term *e-learning* is used.

According to [Tsai and Machado 2002], *e-learning* is defined as follows:

> *"**E-learning** is mostly associated with activities involving computers and interactive networks simultaneously. The computer does not need to be the central element of the activity or provide learning content. However, the computer and the network must hold a significant involvement in the learning activity."*

As the quotation mentions, *e-learning* implies the usage of computers for learning purposes. Concerning web-based learning, which is restricted to deliver the content over the World Wide Web (WWW), *e-learning* does not specify the transmission method. Online learning is connected to available learning materials in a computer environment, while not demanding a network. Distance learning is the "oldest" term and does not require the use of computers or networks. Distance learning includes the interaction between learners or students within a class over a distance for example, receiving the course materials by mail and learning at home. [Tsai and Machado 2002]

This work is located in the field of *e-learning*. Thus, only this term is used.

2.3.3 Adaptive Systems

An *adaptive system* adapts itself or another system to various circumstances. The process of adaptation is based on user's goals and preferences. These properties of the user are stored in a user model. The user model is hold by the system and provides information about the user like for example, knowledge, goals, etc. A user model gives the possibility to distinguish between users and provides the system with the ability to tailor its reaction depending on the model of the user. [Brusilovsky and Maybury 2002]

In the context of e-learning, *adaptive systems* are more specialized and focus on the adaptation of learning content and the presentation of this content. According to [Mödritscher *et al.* 2004], an adaptive system focuses on how the knowledge is learned by the student and pays attention to learning activities, cognitive structures and the context of the learning material.

In Figure 2.2, the structure of an *adaptive system*, according to [Brusilovsky and Maybury 2002], is shown. The system intervenes at three stages during the process of adaptation. It controls the process of collecting data about the user, the process of building up the user model (user modeling) and during the adaptation process.

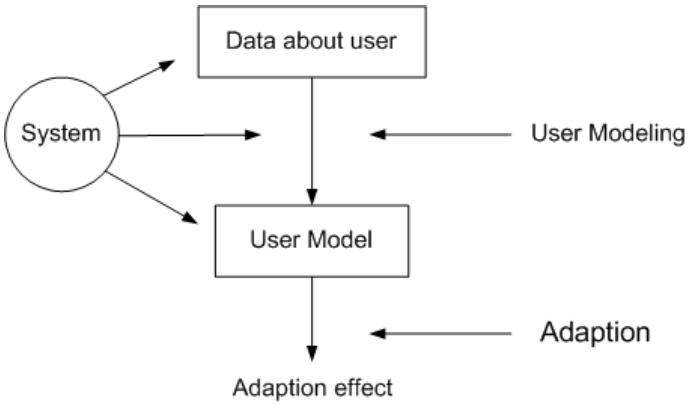

Figure 2.2: The Structure of an Adaptive System [Brusilovsky and Maybury 2002]

Beside this structure of an adaptive system, there exist several other models. [Weibelzahl 2003] lists the Benyon and Murray's model, the Oppermann's model and the Jameson's model.

An adaptive system for e-learning is called an adaptive e-learning system. This restricts the purpose of an adaptive system to the field of e-learning.

An adaptive e-learning system is described, according to [Stoyanov and Kirschner 2004], as follows:

> *"An adaptive e-learning system is an interactive system that personalizes and adapts e-learning content, pedagogical models, and interactions between participants in the environment to meet the individual needs and preferences of users if and when they arise."*

Thus, an *adaptive e-learning system* takes all properties of adaptive systems. To fit the needs for the application in the field of e-learning, *adaptive e-learning systems* adapt the learning material by using user models.

In the following section *adaptive e-learning systems* are described in more detail.

2.4 Adaptive E-learning Systems

As already depicted in Section 2.2.2, particular information about the user is needed to change the behavior of the system in order to satisfy the needs of that user. In

adaptive systems this information is stored in a profile or in a model of the user. Hence, a detailed user profile or a user model is needed to enable adaptivity of the system.

In the context of e-learning, the adaptation of instruction is affected. Instruction is the the form how learner are educated. There exist several possibilities how the instruction is adapted. In the following section (Sub-section 2.4.1), the four main theoretical approaches are described, namely the *macro-adaptive* approach, the *aptitude-treatment interaction* approach, the *micro-adaptive* approach and the *constructivistic-collaborative* approach. Different types of systems, with its relation to these theoretical approaches are described in Sub-section 2.4.2.

2.4.1 Theoretical Approaches

Theoretical approaches describe the different possibilities of adaptive instruction. Since adaptive instruction has a history of more than 100 years, the approaches are listed in chronological order beginning with the oldest approach.

These approaches are applied by the different types of adaptive e-learning systems, which are introduced in Sub-section 2.4.2.

Macro-adaptive Approach

Early attempts to personalize instruction to learners took place on the so-called macro-level. The students were grouped or classified by grades. This grouping resulted in a homogeneous evaluation of the learners and had minimal effects an the adaptation because the groups received different instructions very seldom. To better accommodate different student abilities, the *macro-adaptive* approach was invented in the early twentieth century, where the adaptation of instruction is concerned on a macro-level as well. Within the *macro-adaptive* approach, alternative instructions are computed, based on a few main components such as learning objectives, levels of detail and delivery system. The selection of the appropriate instruction is mostly based on the student's instructional goals, general abilities and achievement levels in the curriculum structure. [Park and Lee 2003]

According to [Corno and Snow 1986], the selection of instructions (i.e., activities) depends on learning objectives such as compensate students' weaknesses or developing new skills and student aptitudes. These aptitudes are categorized into three types, namely intellectual abilities and prior achievement, cognitive and learning styles and academic motivation and personality.

Aptitude-treatment Interaction Approach

The aptitude-treatment interaction (ATI) approach adapts instructional strategies to students aptitudes. This strategy recommends different types of instructions for students with different characteristics. [Mödritscher *et al.* 2004] lists the most important characteristics as intellectual abilities, cognitive styles, learning styles, prior knowledge, anxiety, achievement motivation, and self-efficiency.

ATI further offers the user full or partial control over the learning process. The user is able to control the style of the instruction or the way through the course. Three levels of control are defined, complete independence, partial control within a given task scenario and fixed tasks with control of pace. Studies have shown that the learner's aptitudes influence the learning result when offering different levels of control of the instruction to the learner. For example, students with low prior domain knowledge get better results if this control is limited. [Corno and Snow 1986]

Micro-adaptive Approach

Learning needs during instruction are used by the micro-adaptive approach to adapt the instruction. Theses needs are examined and an appropriate prescription is generated. Compared to the pretask measurements of the macro-adaptive and the ATI approach, the micro-adaptive approach is rather based on on-task measurements. The student behavior and performance are observed by measuring e.g., response errors, response latencies and emotional states.

The first model for the micro adaptive approach is the idea of programmed instructions and was originally applied by Pressey in the year 1926. Through the usage of technology, a number of different micro-adaptive instructional models have been developed. These models differ from the programmed instruction idea by applying a specific model or learning theory. [Park and Lee 2003] lists following existing models: the mathematical model, the trajectory model, the Bayesian probability model and the structural and algorithmic approach.

According to [Mödritscher *et al.* 2004], in case of the micro-adaptive approach adaptive e-learning is separated in two main processes, the diagnostic process and the prescriptive process. The first step (the diagnostic process) is used to characterize the learner by identifying the aptitudes or the prior knowledge and to formulate the task. Secondly, the interaction between the learner and the task is optimized by adapting the learning content to the students aptitudes and actual performance.

Constructivistic-collaborative Approach

The constructivistic pedagogical approach focuses on how an e-learning system can be integrated into the learning process. The learner takes an active role in the process of learning, where the knowledge is constructed by experiences in the specific knowledge domain according to the constructivistic learning theory.

Another major part of this approach is the employment of collaborative technologies, where the pedagogical approach of collaborative learning activities is integrated. Five characteristics of effective collaborative learning are identified by [Soller 2001], namely participation, social behavior, performance analysis, group processing and conversation skills and primitive interaction. To enable a learning success through collaborative technologies, these five characteristic should be available to the learner.

2.4.2 Types of Systems

This section describes types of systems with the help of the theoretical approaches introduced in Sub-section 2.4.1. Starting with macro-adaptive systems, intelligent tutoring systems and adaptive hypermedia system are presented.

Macro-adaptive Instructional Systems

As already mentioned in Sub-section 2.4.1, the macro-adaptive is the oldest approach where students were simply tracked by grades of ability tests. Macro-adaptive instructional systems where developed to tailor the instruction to the learner's abilities. [Park and Lee 2003] mentions the Burke plan, Dalton plan and Winnetka plan as early systems applying the macro-adaptive approach. Within these systems the students were able to go through the learning material at their own pace.

In 1963, the Keller plan was developed at the Columbia University. The Keller plan is a macro-adaptive system where the instructional process was personalized for each student [Mödritscher *et al.* 2004]. It was the first macro-adaptive system used at many colleges and universities all over the world. Until around 1985 several other macro-adaptive instructional systems were developed.

The examples macro-adaptive instructional systems given so far should demonstrate the history of adaptive e-learning and its application. These systems were applied in many schools and universities by providing only weak adaptation.

Computer-managed Instructional Systems (CMI)

An exceptional position take the Computer-managed Instructional Systems (CMI). CMI systems provide many macro-adaptive instructional features offering the instructor possibilities to monitor and control the learning activities of the student. Further, CMI systems integrate features of micro-adaptive models (e.g., prediction of student learning needs). This makes CMI systems more effective concerning adaptive e-learning compared to pure macro-adaptive systems. [Park and Lee 2003]

Intelligent Tutoring Systems

Intelligent Tutoring Systems (ITS) are adaptive instructional systems applying artificial intelligence (AI) techniques. The goal of ITS is to provide the benefits of one-on-one instruction automatically and cost effectively [Shute and Psotka 1996]. As in other instructional systems, ITS consist of components representing the learning content, teaching and instructional strategies as well as mechanisms to understand what the student does or does not know. In ITS these components are arranged into the *expertise module*, the *student-modeling module*, the *tutoring module* and a *user interface module* (see Figure 2.3) [Brusilovsky 1994]. The *expertise module* evaluates the performance of the student and generates instructional content. The *student-modeling module* represents the user's current knowledge and estimates his reasoning strategies and conceptions. This information is used by the ITS to determine, how the teaching process should continue. The *tutoring module* holds information for the selection of instructional material. This information describes how this material

should be presented and when. The *user interface module* is the communication component that controls interaction between the student and the system.

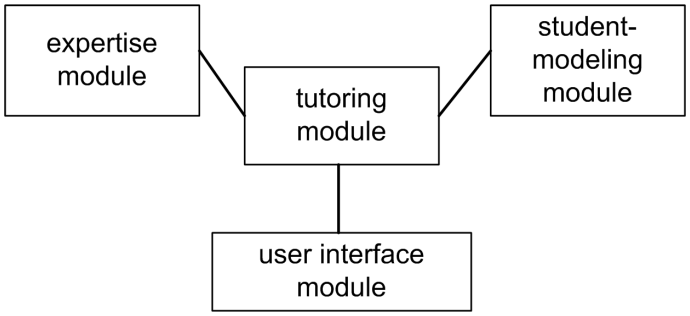

Figure 2.3: Components of an ITS [Brusilovsky 1994]

ITS apply the micro-adaptive model since the decision about learning diagnosis and instructional prescriptions are generated during the task. Further, the combination with aptitude variables allows the expertise module to generate conditions for instructions based on the learner's characteristics. [Mödritscher *et al.* 2004]

A variety of AI techniques are used to represent the learning and teaching process. For example, some ITS systems capture topic related expertise in rules. This enables the ITS to generate problems on the fly, combine and apply rules to solve the problems, assess each learner's understanding by comparing the software's reasoning with them, and demonstrate the software's solutions to the participants. The development of an expert system that provides comprehensive coverage of the subject material remains the major problem for ITS.

Adaptive Hypermedia Systems

The development of Adaptive Hypermedia Systems (AHS) can be traced back to the early 1990s. The hypermedia model was extended by utilizing user models. AHS are inspired by ITS and try to combine adaptive instructional systems and hypermedia-based systems. [Brusilovsky 1996] describes the definition of AHS as follows:

> *"By adaptive hypermedia systems we mean all hypertext and hypermedia systems which reflect some features of the user in the user model and apply this model to adapt various visible aspects of the system to the user."*

Thus, three criteria are satisfied by AHS. First, the system is based on hypertext or hypermedia. Second, a user model is applied and third, the system is able to adapt the hypermedia by using this user model. So far, AHS have been used in educational systems, e-commerce applications, information systems and help systems.

[Brusilovsky 2001] distinguishes between two different types of AHS regarding adaptation methods. The first group, which deals with *adaptive presentation*, provides an adaptation of the content, that can be presented in different ways or orders.

The content can be adapted to various details, difficulty, and media usage to satisfy users with different needs, background knowledge, interaction style and cognitive characteristics. Adaptation of the navigation is provided by the so called *adaptive navigation support* group. It can be implemented as direct guidance, adaptive hiding or re-ordering of links, link annotation, map adaptation, link disabling and link removal. [Kinshuk and Lin 2003]

The introduction of hypermedia and the Web has had a great impact on adaptive instructional systems but there are some limitations of AHS. According to [Park and Lee 2003], there was little empirical evidence for the effectiveness of AHS. [De Bra 2000] states, that if prerequisite relationships are omitted or are just wrong, the user may be directed to pages that cannot be understood by him because of necessary prior knowledge in this domain. Another drawback is that the same page might look different if this page is visited again. When the document is adapted to a developing user model, each time a user visits a particular page again, it may look different. This can cause confusion and loss of orientation for the user.

[De Bra 2000] concludes, that AHS has the potential to provide the users with freedom regarding the navigation through the instruction. Further the users can ensure that the presented learning material is relevant and can be understood by him.

Adaptive Educational Hypermedia Systems

A subtype of AHS are the adaptive educational hypermedia systems (AEHS). As implied in the name, AEHS are applied in the context of education. This type of systems is based on the AHS. The hyperspace of AEHS is kept very small since the documents are related to a specific topic. The focus of the user modeling is on the domain knowledge of the learner. [Brusilovsky 1996]

According to [Henze and Nejdl 2003], an AEHS consists of a *document space*, a *user model*, *observations* and an *adaptation component*. The *document space* belongs to the hypermedia system and is enriched with associated information (for example annotations, domain graphs or knowledge graphs). The *user model* stores, describes and infers information, knowledge and preferences about a user. *Observations* represent the information about the interaction between user and AEHS. These *observations* are used for updating the *user model*. Rules for adaptive functionality (if for example a document should be suggested for learning or to generate learning paths) and adaptive treatment (arrange links to further documents depending on the needs of a particular user) are provided by the *adaptation component*.

2.5 Summary

To be able to react in the way expected by the user, a system needs to have an idea about the user's expectations and preferences. To provide a system with this information, at least a user profile is needed to gather this "raw data". To answer to more particular questions, the system needs the raw data stored within a user profile is not enough. The user has to be modeled in more detail. This task is accomplished

by a user model. A user model stores semantically enriched information about the user, restricted to clearly defined areas and domains. For example, a user model provides information about the user's learning styles, educational curriculum, history of interactions with the system and the domain knowledge. This allows adaptive e-learning systems to adapt the learning content to a specific user.

Concerning adaptive e-learning systems, they are classified by referring to the used theoretical approach or by the type of system. This section distinguishes between four theoretical approaches, namely macro-adaptive approach, aptitude-treatment interaction approach, micro-adaptive approach and the constructivistic-collaboration approach. The macro-adaptive approach is the oldest and can be traced back to the early 1900s. Providing adaptation only at a macro-level, the instruction is not very different for different learners. Aptitudes of the learner are considered by the aptitude-treatment interaction approach. The learner's characteristics define the type of instruction for a particular learner. Needs of the learner are used to adapt the instruction by the micro-adaptive approach. These needs are identified during the task by on-task measurements. The constructivistic-collaboration approach applies the topical constructivistic pedagogical approach combined with the advantages of collaborative learning.

The second classification is done by considering three types of the adaptive e-learning systems. First, macro-adaptive instructional systems, which were popular in the 60s until the early 80s. Second, ITS apply the micro-adaptive approach in combination with the aptitude-treatment interaction approach and utilize AI techniques to represent the learning and teaching process. The newest type of adaptive e-learning systems are the AHS which were invented in the early 1990s. AHS were inspired by ITS and try to combine adaptive instructional systems with hypermedia-based systems. They provide two types of adaptation: the content-level adaptation and the link-level adaptation. At content-level the presentation of the learning material is adapted to the needs stored within the user model. Hypermedia systems make it possible to navigate through documents by following links. This property is used by AHS and AEHS where these links are adapted (e.g. hide, reorder or delete links).

Every personalized system including adaptive e-learning systems base their adaptation processes on user models. In the following Chapter 3, user models are depicted in more detail.

3. User Modeling in Adaptive Systems

3.1 Introduction

In general, the adaptation process can be described by three stages: retrieving the information about the user, processing the information to initialize and update a user model, and using the user model to provide the adaptation. Beside the stated difference between a user model and a user profile in Section 2.2.1, within this chapter only the term user model is used because user profiling is simply seen as the process of collecting raw data about the user. The literature related to the topic of this chapter uses the terms user profile and user model often as synonyms.

In the process of adaptation there can be distinguished between two different personas. At first the learner or student with its goal to acquire knowledge and second the teacher. The goal of a teacher is to mediate the covered knowledge of a course to the learners. Therefore, both points of view must be present in an e-learning system.

The user model is an essential component in adaptive e-learning systems. The adaptation of an e-learning system mainly involves choosing and presenting each successive teaching activity as a function of entire scope of learner's knowledge of the subject being taught and other relevant features of the learner, which are in turn maintained in a learner model. Therefore, the learner model is used to modify the interaction between system and student to suit the needs of individual students.

This chapter, at first describes the role of user modeling within adaptive systems in general (see Section 3.2). To focus on adaptive e-learning systems, the different types of user models and their usage in adaptive e-learning systems are introduced in Section 3.3. These two sections give an introduction into the topic of user modeling within adaptive systems.

Focusing on learner modeling, Section 3.4 depicts different approaches of learner models followed by a description of available modeling techniques to build up these models (see Section 3.5). At the end of this chapter, a summarization is given (see Section 3.6).

3.2 The Role of User Models in Adaptive Systems

As already stated in Section 2.1, user modeling takes a lot of efforts. This section describes the role of a user model within an adaptive system. According to [Kay 2000b], there are three main ways a user model can assist in adaptation. These are shown in Figure 3.1 indicated by arrows. Between the small rectangles the interaction between the user and the system is delimited.

The *interpret user actions* arrow represents user actions at the interface. This arrow covers all possible actions, which are available through the user interface, such as mouse actions, typing at the keyboard and audio/video input. A user model can support the system in interpreting this information. The figure mentions the example when the user input is ambiguous. In this case, the user model might

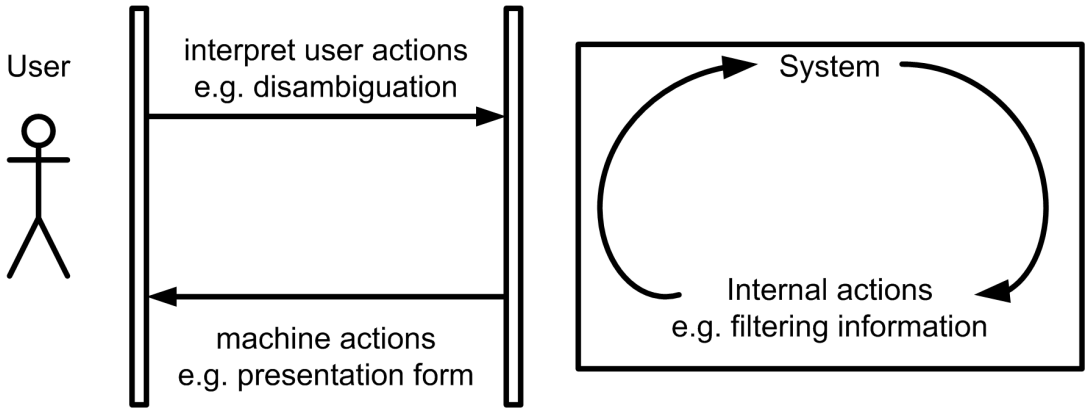

Figure 3.1: Role of the User Model in Adaptation [Kay 2000b]

support or even enable the system to clarify the user input. Further, the user model can help the system to interpret incorrect user actions. [Kay 2000b] lists applications in fields like natural language understanding, command line interpreters, spelling errors by dyslexic users or typing problems by users with motor difficulties.

The *machine actions* arrow represents the actions initiated by the system. A user model can be utilized to control and modify these actions to the preferences of the user. This method includes tailoring the system's behavior and look to the user or adaptation of the content as well as the presentation of the content. For example, e-learning systems may adapt their actions to the user's domain knowledge with simpler information for students with less knowledge and more difficult material for students with more knowledge. Adaptive hypermedia systems, which are described in Section 2.4.2, focus on the adaptation of navigation and content considering the preferences and domain knowledge stored in the user model.

The third utilization of a user model for adaptation takes place inside the system. The user model supports the system during *internal actions*. Often, *internal actions* are filtering processes where the received information is sieved.

A combination of these action is often used for adaptive systems. For example, an e-learning system monitors the attempts of the user to solve a given task by using the user model to interpret the user actions. The system might then perform some *internal actions* to select the best suitable instruction. These *internal actions* are influenced by the user model, especially by the user's knowledge and learning preferences. After this internal process, the system generates an action at the interface. The form of the presentation of this action is affected also by the user model. [Kay 2000b]

[Koch 2000] lists seven purposes of a user model:

- Assist a user during learning of a given topic.
- Offer information adjusted to the user.
- Adapt the interface to the user.

- Help a user to find information.
- Give the user feedback about his knowledge.
- Support collaborative work.
- Give assistance in the use of the system.

These purposes are quite user centered and do not explicitly explain the points where the user model influences the system. However, these purposes can be fulfilled by the three possible ways, a user model can influence the adaptations, which have been previously described.

This section described the usage of user models in general adaptive systems. In the following section, the different types of applied user models in adaptive systems are depicted.

3.3 User Models in Adaptive E-learning Systems

User models can be used in very different ways, regarding what is seen as a user of the system. Since there exist different types of adaptive e-learning systems, the applied user models are different. These user models and their usage within the system are described within this section by examining macro adaptive systems, intelligent tutoring systems and adaptive educational hypermedia systems (see Sub-section 3.3.1 - 3.3.3).

In many systems, the user model is not explicitly described as a simple functional module. It can be spread over several components of the system. Thus, it is not clearly visible what is connected to the user model. This fact is considered in the following section. Therefore a user model might not be available as an extra component but properties, which are connected to a user model, are assigned to the term user model.

3.3.1 User Models in Macro-adaptive Systems

In macro-adaptive systems (see Sub-section 2.4.2), the adaptation of instructions is processed on a macro-level of personalization. Instructional decisions are based on learner traits gathered from self-reports, questionnaires and pretests. For macro-adaptive systems it is not necessary to know the domain knowledge during the instruction. The instructional objectives are determined for each student based on his academical profile, previous achievements, and other aptitude and motivation data. The gained knowledge is generated by comparing the results from the pretest and the test after the course. [Park and Lee 2003]

To accommodate different learning abilities and styles of a student, the necessary guidance for each student is determined and alternative instructional materials are selected. The used user models for the learners are not affected or updated during instruction. No activities or occurred problems are stored.

3.3.2 User Models in ITS

An ITS consists, as described in Sub-section 2.4.2, of basically four components (see Figure 3.2). Three of these four components contain a specific user model, namely the expertise module, the student module and the tutoring module. The expertise module contains an *expert model*, the student module contains a *student model* and the tutoring module contains a *pedagogical model* or tutor model.

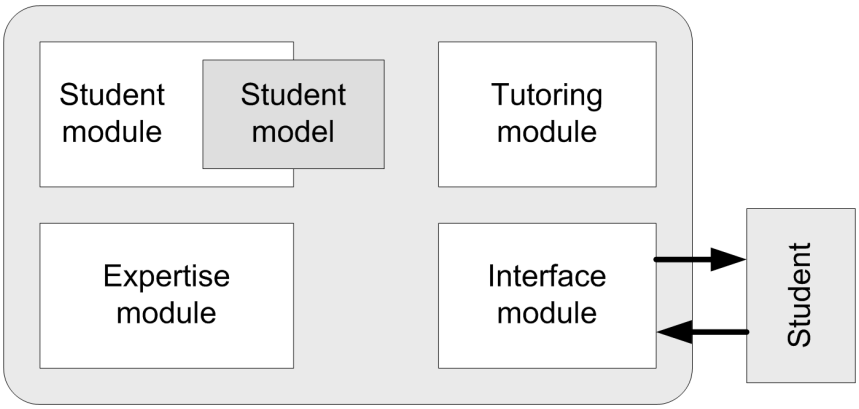

Figure 3.2: Components of an ITS [Brusilovsky 1994]

The *expert model* embeds the necessary knowledge about the domain to provide adaptive feedback, answers to questions and support in problem solving. The system's belief about the learner's knowledge is represented by the *student model*. Further, the student model may include characteristics and preferences of the learner. The *pedagogical model* contains rules, which allow the system to teach like a tutor. These rules take properties of the student stored in the student model and use the expert model to calculate the appropriate instruction for the current student.

With this pedagogical model, the tutoring module is able to perform similar functions as a human tutor in conventional training. These functions are for example:

- knowledge diagnostics
- strategic functions
- prediction functions
- assessment functions
- knowledge development
- error remediation
- domain content representation
- exploration space control

The student model is used to adapt these functions to the needs of each student. *Knowledge diagnostics* represent an accurate measurement of the current student's knowledge state. A selection of global teaching plans or teaching strategies are assigned to *strategic functions*. Learning path and learning behavior of the student are covered by *prediction functions* while *assessment functions* cover students assessments and ITS assessments. *Knowledge development*, *error remediation* and *exploration space control* are described in more detail in the following sub-sections. [Brusilovsky 1994; Kinshuk and Lin 2003]

Knowledge Development

To help a student during the knowledge development, an ITS provides four stages. These stages are *what to teach*, *when to teach*, *how to teach* and finally the *implementation of teaching actions*. [Han 2001]

What to Teach: In this stage the student model provides information about the current knowledge of the student. The system is interested in the student's lacks of knowledge. With this information, the tutoring module is able to choose the *curriculum sequence*, to provide *active help and feedback* during the problem solving process or to provide *passive help* on request.

Curriculum sequence is used to provide the student with the most suitable learning sequence. The learning sequence consists of the series of knowledge units to learn and tasks like examples, questions and problems to solve. The *curriculum sequence* helps the student finding a proper path through the learning materials. There are two subtypes for *curriculum sequence*: *high-level sequencing* and *low-level sequencing*. *High-level sequencing* or knowledge sequencing uses the student model and the domain knowledge to select the successive teaching concept or topic. The *low-level sequencing* uses only the student model to determine the following learning task (e.g. example, test or question). [Brusilovsky 1998]

By using recorded student behaviors the system is able to provide *active help and feedback*. Intelligent feedback to solutions of the student, such as error feedback or comparing the student's solution to an example solution, can help to improve the student's comprehension on this topic. Further, additional hints or reminders help the student during the solution process.

Whenever a student requests help for the current instruction, the system provides the student with *passive help* such as hints, answers to questions or extra explanations based on the student's knowledge stored in the student model.

When to Teach: The appropriate moment for knowledge development is calculated by using the student model. This is important when active help is needed during the process of problem solution.

How to Teach: Suitable teaching actions such as explanations, test, examples or problems are chosen by using the student model. These teaching actions are influenced by the learning style or preferences stored in the student model.

Implementation of Teaching Actions: At this stage, some teaching actions can be modified according to the student model. For example, if a student has advanced knowledge in a topic only a small explanation is provided for solving a problem.

Error Remediation

According to [Self 1987], eight remediation methods can be identified, namely *error definition*, *explicit remediation*, *implicit remediation*, *counter examples*, *demonstration of a solution method*, *access to previous experiences*, *repeat attempt* and *tactical retreat*.

Error definition provides a textual description of the error and a recommendation for correction. Explicit presentation of the correct knowledge is given by an *explicit remediation* while *implicit remediation* prompts the correct knowledge or actions and shows hints to the student. *Counter examples* are situations or problems generated by the system. *Access to previous experiences* is given by the user model where these experiences are stored.

Exploration Space Control

The ITS automatically controls the exploration space, while a student is navigating through the domain space. This control takes places in form of limiting information resources, the number of search paths and tools, and the amount of presented information. This controlling concept is used to reduce the student's cognitive load since too much information or possibilities reduce the student's attention and leads to distraction. The control of the exploration space is based on the student's competence level, experience, etc.. [Kashihara *et al.* 2000]

User models in ITS are applied in several components of the system. For the purpose of modeling the user of an ITS the student model is the most important model but the system relies also on relation among the student model, the expertise model and the pedagogical model. To see differences and conjoints between ITS and AEHS, the next section depicts user models in AEHS.

3.3.3 User Models in AEHS

To describe user models in adaptive educational hypermedia systems, at first the possible adaptations must be introduced. According to [Brusilovsky 2001], mainly the presentation of the hypermedia and the navigation through the hyperspace can be adapted. These two ways are subdivided into several adaptive hypermedia technologies (see Figure 3.3).

To be able to tailor the presentation of the learning content and the navigation to the needs of the user, a user model is needed, which includes the user's goals or tasks, knowledge, background and preferences. These properties of a user are used for making adaptation decisions by adaptive hypermedia systems. Additionally, recent user models also store the interests and individual traits. The individual traits include personality, cognitive factors and learning styles, but are not easy to

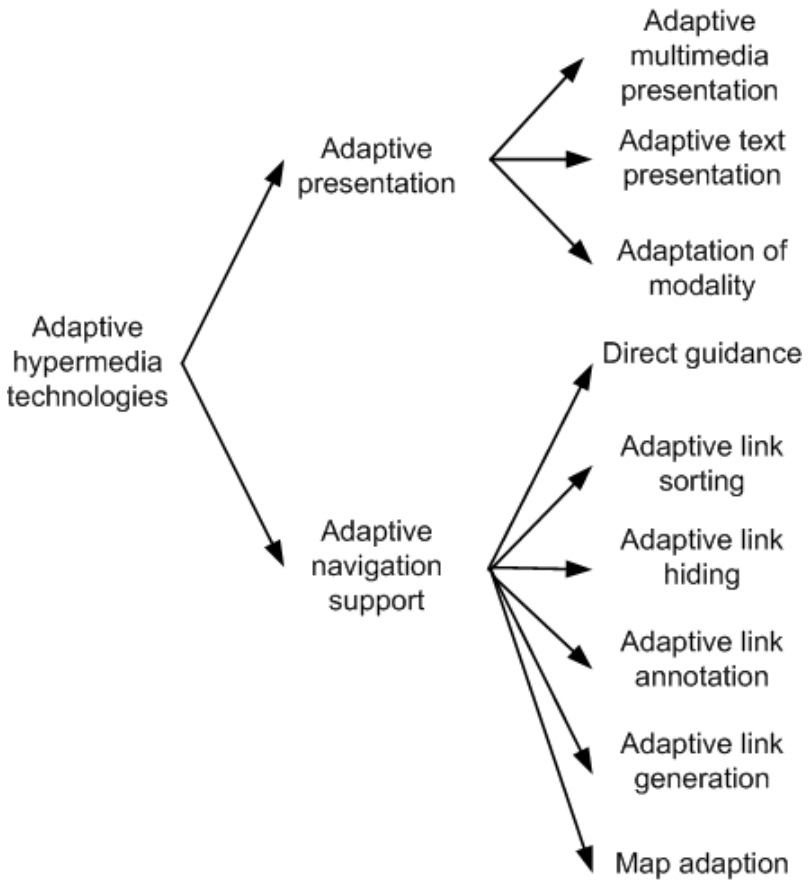

Figure 3.3: Adaptive Hypermedia Techologies [Brusilovsky 2001]

extract. They represent stable features of a user, while interests change over time. [Park and Lee 2003]

[Brusilovsky 2001] describes a new type of adaptation for web-based systems. The nature of web-based systems enables the user to change its learning environment. Thus, several different hardware, software and platforms can be used by the same user. This requires an adaptation to the user's environment and has become an important issue.

According [De Bra *et al.* 1999], an adaptive hypermedia system should consist of a *domain model*, a *user model*, an *adaptation model*, also known as *pedagogical model* and an *adaptive engine*. The *domain model* describes how the information is structured and linked together. The *user model* represents a set of user characteristics and preferences for browsing the hypermedia space and the *adaptation model* consists of pedagogical rules, which define how the *domain model* relates to the *user model* in order to provide adaptation. The *adaptive engine* is responsible for processing the adaptation by manipulating the navigation or content fragments of the page before the adapted page is sent to the user interface.

The models proposed in [De Bra *et al.* 1999] and user models in ITS (see Subsection 3.3.2) are identically. The domain model serves the same purpose as the

expertise model of an ITS. Both models represent the knowledge of the subject domain. The tutor or pedagogical model of an ITS describes pedagogical rules for the instruction, which is comparable to the adaptation model of an AEHS. Finally, the user model of the AHAM reference model is equally in its purpose to the student model of an ITS. To conclude, there are mainly the same user models with the same usage in both types of systems. The reason for this is the historical development, since the development of AEHS is based on ITS, which were invented before AEHS.

The described user models in this section give an overview about the applied models in different types of systems. The preferences and characteristics are mainly modeled within a student or learner model by all systems. Sections 3.2 - 3.3 describe the environment of user models, which roles they have, and which types of user models are used within different types of systems. The terminology of student and learner model is often confusing, although both terms have the same meaning. In the following section, only the term learner model is considered. It describes the learner model in more detail by introducing the different components and the stored information of a learner model.

3.4 Learner Modeling

A learner model represents the system's beliefs about its main target user, the learner, and provides the necessary information for tailoring the instruction to the needs of the learner. This necessary information is represented by the content of a learner model (see Sub-section 3.4.1). The whole content can be grouped regarding different properties of a learner. These groups are arranged in components of a learner model (see Sub-section 3.4.2).

3.4.1 Content of a Learner Model

An extensive learner model must contain information about the learner's domain knowledge, the learner's progress, preferences, goals, interests and other information about the learner, which is important for the used systems. [Self 1994]

[Brusilovsky 1994] states that learner models can be classified according to the nature and form of information contained in the models. Considering the subject domain, the information stored in a learner model can be divided into two major groups: *domain specific information* and *domain independent information*

Domain Specific Information

Domain specific information represents a reflection of the learner's state and the level of knowledge and skills in a particular subject. [Brusilovsky 1994] names the model of the domain specific information *knowledge model*. The knowledge model can be based on different types of models or a combination of these types. Possible types of knowledge models are described in the following.

Scalar Model: Within a scalar model, the level of the learner's knowledge on the entire domain is described by one identifier such as a number in the range from 1 to 5. The scalar model is the simplest form of a knowledge model and provides no information about knowledge in a sub-domain. [Han 2001]

Overlay Model: The entire domain information consists of a set of knowledge elements or curriculum elements and represents the expert's knowledge in this domain. The overlay model describes the learner knowledge as a subset of the complete domain model. Lack of knowledge by the learner is derived by comparing it to the expert's knowledge. To each knowledge element in the learner's overlay model, a certain measure is assigned representing the estimated knowledge of the learner on that element. The measure can be a scalar (for example an integer, a probability measure or a flag) or a vector estimate. [Henze and Nejdl 2003]

A detailed description of the overlay modeling techniques is given in Sub-section 3.5.1.

Error Model: A disadvantage of overlay models is the incapability of storing errors or mistakes made by the learner. For this reason, the bug model or error model has been developed. With an error model, it is possible to define and reflect erroneous behaviors of learners and the reasons for these errors. Error models can be categorized into *perturbation models* and *differential models*.

Perturbations or misconceptions for each knowledge element are stored in a *perturbation model*. It is assumed that one or more perturbations exist for each knowledge element of the domain model. Thus, a learner's perturbation model represents a subset of all possible perturbations, which are the cause for incorrect learner behavior regarding particular knowledge elements. [Brusilovsky 1994]

Genetic Model: The described models, overlay and error model, represent the state of the learner's knowledge. But, these models do not express the structure of the domain knowledge. Therefore, genetic models are used to describe the development of the learner's knowledge. This process can evolve from simple to complex or from special to general. For example the learner starts with a very special knowledge and proceeds toward a broad and general knowledge. It is possible to describe a genetic model by a genetic graph, where the nodes and the relationships between the nodes represent knowledge elements and their interactions. [Han 2001]

Other Domain Specific Information: Besides the already described domain specific information stored by different types of models, additional domain specific information can be stored in the learner model. According to [Han 2001] this information includes:

- prior knowledge about the domain of the learner,
- records or learning activities (taken lectures, number of asks for help, time to solve problems), and
- records of assessments and evaluations.

In general, the additional domain specific information may contain necessary information for specific purposes of the learner model related to subject domains.

Domain Independent Information

In addition to the learner's current knowledge level, domain independent information is needed to enable adaptivity. Domain independent information about a learner may include *learning goals*, *cognitive aptitudes*, *motivational state*, *background and experiences*, *preferences* as well as *factual and historic data*. [Han 2001]

Goals: To establish the correct teaching strategy, it is important to know the learner's goals. These goals answer the questions why the learner uses the e-learning system and what the learner wants to achieve. The goals can be divided into two different types. First, the learning goal, which is relatively stable for a course unit. Second, the problem-solving goal, which may change from one problem to another even within one teaching unit. Example for learning goals are to pass an exam or solve a particular problem.

Cognitive Aptitudes: Cognitive aptitudes are intellectual abilities for differing kinds of cognitive performance. For example, musical aptitude, math aptitude and reading aptitude are all different kinds of cognitive aptitudes.

Motivational States: To measure the drive in instruction the motivational state of the learner is used. The motivation is measured using a number of long-term and short-term parameters. Such parameters are for example effort, attention, interest, distraction, persistence, etc. These parameters are connected to other factors such as knowledge level, readiness, complexity of the topic and learning outcome.

A learner model that considers motivation of a learner and knowledge state, was proposed by [Far and Hashimoto 2000]. Within this learner model, the motivational state is represented by a Bayesian network, where the graph encodes the dependencies among motivational facets and learning activities. For example a path from the distraction property to attention indicates that distraction is influencing the attention of the learner.

Background and Experience: To derive parameters of the learner model, information about background and experiences is used. Background information includes skills that may affect the learning achievement. Such information is for example, profession, work experience or perspectives. [Brusilovsky 1994]

Experience represents knowledge about the learning environment. Learner, which are new to a particular learning environment or even new to e-learning may need different system support regardless if they are novices or experts in the subject domain. This information might be used to select the appropriate adaptive navigation method.

Preferences: The learners may have different preferences related to some aspects of the learning environment. These preferences are considered as not inducible by the system. Thus, the learner has to inform the system directly or indirectly about his or her preferences. It is important for an adaptive e-learning system to present and organize the learning material based on the learner's preferences. Learner preferences can also be used to form groups of learners. This technique is called group modeling, where learners with same preferences form a group (group modeling is described in Sub-section 3.5.1). Two parts of the preferences are the *learning style* and the *multiple intelligence*, whereby *learning style* and *multiple intelligence* are mutually related to each other.

The theory of *multiple intelligence* was first described by Howard Gardner in the year 1983. His latest research indicates that there are eight distinct forms of intelligence: linguistic, logical-mathematical, spatial, kinesthetic, musical, interpersonal, intrapersonal, and naturalist [Lane 2000]. These eight forms of multiple intelligences are described as follows:

- *Linguistic intelligence* is the ability to use words and language. These learners have highly developed auditory skills and are generally elegant speakers.
- *Logical/Mathematical intelligence* is the ability to use reason, logic and numbers. These learners think conceptually in logical and numerical patterns making connections between pieces of information.
- *Spatial intelligence* is the ability to perceive the visual. These learners tend to think in pictures and need to create vivid mental images to retain information.
- *Kinesthetic intelligence* is the ability to control body movements and handle objects skillfully.
- *Musical intelligence* is the ability to produce and appreciate music.
- *Interpersonal intelligence* is the ability to relate and understand others. These learners try to see things from other people's point of view in order to understand how they think and feel.
- *Intrapersonal intelligence* is the ability to self-reflect and be aware of one's inner state of being.

- *Naturalist* describes the ability to recognize and classify numerous species of the environment (flora and fauna).

Gardner stated that everyone possesses all of these eight intelligence but in different degrees. Considering multiple intelligence during the process of adaptation, the learning environment is able to tailor the learning material according to the learners strengths, which allows to hold the learning progress on a maximum base. [Lane 2000]

Learning styles are different approaches or ways of learning. Multiple intelligence determines multiple learning styles. Therefore, every learner has different preferences for how, when, where and how often to learn knowledge. An example for a learning style model is the WAVI model. WAVI stands for *wholist*, *analyst*, *verbalizer* and *imager* and describes preferences about the visualization of learning material. For example, if a learner prefers images then he is covered by the type *imager*. [Lane 2000]

Factual and Historic Data: Demographic data such as name, age, parents, ID etc. is often stored in learner models. This information, combined with other factual data such as for example interests, is necessary to initialize an individual learner model. [Han 2001]

This sub-section described the content of a learner model. The content may simply be subdivided into domain specific and domain independent information. Beside these classifications of information, it is also possible to divide the structure of a learner model into several logical components.

3.4.2 Components of a Learner Model

Before dividing a learner model into components, the stored information about the learner has to be analyzed and grouped regarding different types and levels of information.

The components are strongly connected to the application of the learner model, but it is common to build a *performance* and a *teaching history* model. The *performance model* stores data related to the assessment of the learner's overall skills, as well as data related to the learner's previous knowledge. The *teaching history* model keeps track of the material presented to the learner during instruction and the learner's mastery of instruction units. [Zhou and Evens 1999; Jeremić and Devedžić 2004]

Additional components are necessary to provide a complete information about the learner. [Zhou and Evens 1999] list two additional components, the *reply history* and the *solution record*. A learner *reply history* is attached to each instruction unit and stores information about covered learning material and learner answers during this instruction unit. The amount of errors and the description of these errors, which where made during the process of problem solving are stored in the *solution records*.

Another approach to split the learner model into several components is described by [Castillo *et al.* 2003], where the learner model is divided into three components: a *profile*, a *cognitive overlay model* and a *course overlay model*. The learner *profile*

stores information such as name, age, learning style etc. The system's beliefs about the learner knowledge is recorded in the *cognitive overlay* component while the *course overlay* component supplies information about the learner's interactions with the system.

To describe the technical details about a learner model, additional technical components are used. Technical components are for example a *data reader*, a *data writer* and a *session manager*. The *data reader* and the *data writer* components are responsible for providing access to a data storage. A *session manager* controls and coordinates all other components and is the controlling unit of a learner model. [Jeremić and Devedžić 2004]

A learner model is a combination of all relevant data about the learner with respect to a learning environment. There are common types of information among learner models like the described subject domain information, learning goals, motivation, background and experience, cognitive aptitudes, preferences and demographic data. The content of a learner model is arranged into several components according to the type of information. To gather the necessary information, different techniques are used as described in the following section.

3.5 Modeling Techniques

After identifying the necessary information stored in a learner model, the process of acquiring this information becomes important. There are several methods to construct a learner model, which are described in Sub-section 3.5.1. The construction of a learner model is followed by the initialization of this learner model. It has to be filled with initial data. The process of initialization is depicted in Sub-section 3.5.2. To keep the information stored in the learner model up to date, changed characteristics of the learner has to be determined to fulfill the requirement of representing current aspects of the learner. Besides the update of the stored information in the user model, the delivery of this new data is described in Sub-section 3.5.3.

3.5.1 Methods to Construct Learner Models

Before a user model can be used it has to be constructed. There are many different methods, which can be applied during the construction. This section describes machine learning methods, Bayesian methods, overlay methods, stereotype methods and plan recognition methods.

Machine Learning Methods

Machine learning methods applicable for user modeling include for example rule/tree learning methods, probabilistic learning methods and instance/case-based learning methods. A learner model profits of utilizing machine learning methods by means of increasing accuracy, increasing efficiency and even expandability to where it was not possible to construct a learner model before. [Sison and Shimura 1998]

According to [Webb *et al.* 2001], the purposes for which machine learning methods can be used are to model the cognitive processes that underlie the learner's actions, to model the differences between the learner's skills and expert skills, to model learner's behavioral patterns or preferences and even to model the characteristics of the learner. However, although the first two purposes are addressed very often, modeling behavior or characteristics is still very rare.

Bayesian Methods

Bayesian methods and their applications, like Bayesian networks, are very powerful and versatile. In general, Bayesian methods are related to the machine learning methods (see Sub-section 3.5.1) but Bayesian methods are often used within user models and are therefore described separately.

Bayesian methods support the use of probabilistic inference to update and improve belief values. The main goal of Bayesian networks is to enable probabilistic inference. This section does not describe the Bayesian methods, since this is out of scope of this work, but focuses on their application in the field of user modeling.

According to [Li and Ji 2005], Bayesian networks are used for plan recognition (see Sub-section 3.5.1), user's needs inference and affective state assessments. To infer the current state and needs of the learner, taken pauses and errors are considered. Further, goals and needs are inferred by using the learner's background, actions and queries.

The current emotional and mental aspects of the learner are an important indication of the learner's state, intention and needs [Li and Ji 2005]. Therefore, the affective state is a point of interest and can be generated by using Bayesian networks. For example, the emotional states are modeled as consequences of how the current action fits to the learner's goals and preferences.

Overlay methods

The overlay approach, which was introduced by Carr and Goldstein 1977, is based on the aspect, that the learner model is a subset of the expert model. The expert model is subdivided into several smaller parts and modularized into specific topics or concepts. Each of these small parts can be connected to a particular learner model. [Sison and Shimura 1998]

Overlay models are widely used in user modeling systems, where they are applied to model the educational domain. The domain knowledge of the learner is represented as a subset of the system's expert domain knowledge. The knowledge of the learner is constructed on a concept-by-concept base and updated as the user proceeds through the course. [Brusilovsky 1996] states that an overlay model allows a flexible model of the learner's knowledge for each topic.

The complexity of an overlay model depends on the structure of the domain knowledge, where the granularity is important. Further, the estimation of the learner's knowledge is important and is measured by examining the sections the learner has read and the tests the learner has taken. [Conlan *et al.* 2002a]

Stereotype methods

A stereotype is a collection of frequently occurring characteristics of users. Creating stereotypes is a very common way of user modeling. New learners are categorized and assigned to a stereotype according to their initial user model characteristics. The small amount of initial information is used to infer a large number of default assumptions. When more information about individual assumptions becomes available the default assumptions are altered. [Rich 1979]

There are two types of stereotyping: *fixed* and *default*. In *fixed* stereotyping learners are cast according to their performance into a predefined stereotype that is determined by an academic level. *Default* stereotyping is a much more flexible approach. At the beginning of a session the learners are stereotyped to default values, but as the learning process proceeds and learner performance data is obtained, the settings of the initial stereotype are gradually replaced by more individualized settings. [Kay 2000a]

There are three important elements in a stereotype. First, *triggers* are used to activate a stereotype. Without defined *triggers*, it is not possible to assign a new learner to a particular stereotype. For example, if the learner is a novice Linux user, the trigger "no prior knowledge about Linux" is activated. Since the "no prior knowledge about Linux" trigger is related to the novice-stereotype the novice-stereotype is assigned to this user. The second element of a stereotype is the associated information *inferences*. Activating the novice-stereotype for a learner would infer all the defined aspects, which a novice user is assumed to know. Third, *retraction* conditions. There might be a retraction facility to deactivate a stereotype. This can be important when essential *triggers* change its state. For example, if the novice Linux user gets experiences over time, the "no prior knowledge about Linux" trigger gets deactivated and therefore the novice-stereotype is no longer valid for this user. [Kay 2000b]

The problem using stereotypes is the work to construct and fill appropriate stereotypes. According to [Kobsa 1993], the developer of a user modeling system has to think about tasks like *user subgroup identification*, *identification of key characteristics* and *representation in hierarchically ordered stereotypes*. Subgroups subdivide the community of all possible users of the system. After the identification of subgroups and their key characteristics, a stereotype is assigned to each subgroup.

To identify user subgroups, the user model developer must think about different groups within an expected user population. The members of a subgroup should own certain application relevant characteristics. Thus, to identify stereotypes, knowledge about the application domain is needed. This restricts the usage of a particular stereotype to an application domain.

Further, key characteristics or triggers must be identified in order to be able to determine the relevancy of a learner to a specific subgroup. The amount of these characteristics should be kept very small to provide a fast and easy assignment to a subgroup.

After all subgroups are identified and related characteristics are assigned, a representation of this structure is constructed. The collection of all represented char-

acteristics of a subgroup is called a stereotype for this subgroup. If a part of the content of a stereotype is the content of another stereotype it is possible to build up a hierarchical structure of all stereotypes. [Kobsa 1993]

It is also possible to form stereotypes based on the background knowledge of the learner. Mostly a linear hierarchy of stereotypes like beginners, intermediates and experts is used.

Another modeling technique, which is similar to stereotype modeling is *group* or *community modeling*. The community approach was implemented in Doppelgänger [Orwant 1993], where communities represent a group of learners. The difference to stereotype modeling is that a learner is classified by belonging to several communities. If a user model has no explicit information about a particular characteristic the value is retrieved from the communities the user belongs to.

Plan Recognition

Plans are used to describe the learner's intentions and desires, where a plan is a sequence of learner actions that achieve a certain goal. Plan recognition is based on observing the learner's input actions. Such systems try to determine all possible learner plans, which are valid concerning the observed actions. This calculated set of plans can be decreased by taking new leaner actions into consideration. [Li and Ji 2005]

According to [Kobsa 1993], there are basically two kinds of techniques used to recognize the learner's plan. In the first approach *plan libraries* are constructed. A plan library contains all possible plans and the selection of the actual plan is based on the observed actions by matching these actions to the set of plans. The problem with this technique is that all allowed sequences of learner actions have to be stored within a plan. This demands a lot of computational work beforehand and a huge storage for the plan library.

The second approach is called *plan construction*, where the system controls a library of all possible learner actions combined with the effects and the preconditions of these actions. The sequence of learner actions is enriched by all allowed succeeding user actions. Possible next user actions are calculated by comparing the effects of preceding actions with the preconditions of actions stored in the actions library. [Kobsa 1993]

In general, the plan recognition method is limited by the requirement that all possible learner plans have to be specified before. This need not be a problem if the domain is small enough in which the learner can only follow a limited number of goals.

Besides the described methods, learner modeling systems often apply specifically developed methods to construct the model. For example, the learner's experience can be calculated from their navigational actions or from considering the time spent on pages. Further, special rules are implemented to conclude properties or behavioral concepts of the learner.

The process of constructing a model is in most cases not based on one method. Rather a combination of several methods is applied. Especially stereotyping and overlay methods are utilized.

After the construction of a learner model by applying one or a combination of the described methods, in most cases it has to be filled with information and data about the learner. This process is called initialization. Examples where no learner model initialization is performed are generic user models. Generic user models are constructed without concerns about the application field. Therefore, it is not possible or desired to initialize the user model. The initialization process of learner models is described in the following section.

3.5.2 Initialization of Learner Models

The initialization of a learner model represents the process of gathering information about the learner and transferring this information into the model. The initialization process is also problem in the field of recommender systems, where it is known as ramp-up or cold start problem.

This section describes methods, how information about the learner is retrieved. According to [Self 1994], a learner model can be initialized in three ways, through *explicit questions*, *initial testing* or by *stereotyping*.

Explicit Questions

The initial learner models are often constructed by directly questioning the learner. This method is a very effective way to obtain general information about a learner. The problem is to find the appropriate amount of questions and to get the optimum amount of information out of these questions on the other side, too many initial questions could irritate the learner and increase the declination to the system. The worst case would be, if the learner leaves the system and never returns on the other side, too less or not well selected questions do not allow the system to extract enough information to initialize the learner model. [Tsiriga and Virvou 2003]

An alternative that would reduce the number of questions is to use adaptive questionnaires like in [Kurhila *et al.* 2001]. They apply adaptive methods to optimize the questionnaire length by dropping out uninformative questions using Bayesian methods.

Initial Tests

By asking the learner to take a test, the initial parameters in the learner model can be obtained by analyzing the test results. In order to control the length of the test, the concept of neighborhood of knowledge states may be applied. For example, if curriculum elements A and B are in the same neighborhood, mastery of A implies mastery of B. This leads to a reduction of the test length but presupposes a well constructed test. Initial tests are often used to get information about the domain knowledge of the learner. [Self 1994]

Stereotyping

The learner modeling system may use stereotype methods in order to group similar learners to categories (see Sub-section 3.5.1). Although stereotyping is very powerful in providing considerable information based on only a few observations, it does not provide an accurate learner model. The required information to be able to apply stereotyping can be retrieved by using explicit questionnaires. Another method is to assign a new and unknown learner to a default stereotype and refine the applied stereotype by observing the learner. This can also help to reduce the initial questions. [Han 2001]

After filling the user model with information, keeping this information topical becomes important.

3.5.3 Update of Learner Models

Updating a learner model means to bring the contained data and information about a learner up-to-date. Since dynamic and short-term learner characteristics are not constant properties, a change over time has to be considered by the learner model. For the process of updating a learner model, information sources and update methods are needed.

Information Used to Update Learner Models

The used information to update a learner model can be retrieved from different information sources. At first, the information currently stored in the learner model must be considered. This information can be used as a base to infer new information or perform changes on the inferred information. Further, information currently stored in other system components can be of use. For example the domain model of an ITS. The main source of information can be gained through monitoring the learner's interaction with the system.

According to [Kinshuk 1996], there are several ways to obtain information from the mentioned information sources:

- implicit,
- explicit,
- structural, and
- historical acquisition.

Implicit acquisition of information is based on observing actions of the learner during the learning process. Considering direct dialogues between system and learner leads to *explicit* acquisition (e.g. explicit questionnaire). *Structural* acquisition is performed by analyzing interrelations between curriculum elements. For example, if curriculum element A is a prerequisite of element B, an expertise in B implies the mastery of A. Assumptions based on the learner's experience are performed during a *historical* acquisition of information.

Considering the information available to update a learner model there exist different methods how this information is used to update the learner model.

Methods to Update Learner Models

Information to update the learner model has to be derived by analyzing learner responses, the processes of problem solution and learner actions. These three methods are analytical processes and are called cognitive diagnosis. Cognitive diagnoses is defined as the process of inferring a person's cognitive state based on the performance of this person. Another method to update a learner model is to determine old data and do not using this old data anymore. [Self 1993]

Analysis of Learner Responses: The analysis of learner responses is also called *performance measuring* [Brusilovsky 1994]. Basically, questions of an exam during instruction can be divided into simple questions and complex questions. Simple questions are only related to one specific curriculum element while complex questions require the knowledge of more than one single curriculum element. Accordingly, the learner responses to these two types of question must be handled differently. For example, a correct answer to a simple question increases the relevance of the related curriculum element, while a wrong answer decreases the relevance of the underlying curriculum element.

Analyzing the response to a complex question needs more effort. Correct answers may lead to an increase of all related curriculum elements but an incorrect answer needs to be investigated more thoroughly. The question has to be split based on the structure of the domain model and the resulting parts have to be considered. Thus, some parts of the answer maybe correct while others are not. By applying a perturbation or an error model (see Sub-section 3.4.1) the perturbations, which are relevant for the incorrect answer to the question must be determined. This changes have to be considered and an update of the affected properties of the learner model has to be performed.

Analysis of the Process of Problem Solution: Analysis of the problem solving process requires a technology, where all possible correct rules, which can be used by the learner during the solution process, are available. By combining these rules with a collection of misconceptions responsible for error that may occur, the system is able to calculate and detect all correct solution steps and misconceptions made by the learner in every step of the problem solving process. [Brusilovsky 1994]

Analysis of Learner Actions: Actions of the learner can be analyzed by considering them as results of the acquisition of a set of curriculum elements or misconceptions. This is possible if the subject domain is known. For this, a simply tracing of the learner actions is needed. This method is called *issue tracing*. [Brusilovsky 1994]

Discounting Old Data: Considering only topical data reduces the value of old data in the learner model. This gives importance to data, which is derived from recent actions. The process of discounting old data is based on the assumption that the time elapsed since this old data was stored, decreases the importance and the influence of the old data to the current state of the learner. [Webb and Kuzmycz 1998]

Modeling techniques are an important topic during the construction process of a learner model. It is common that a learner model uses a combination of several modeling techniques, especially a combination of stereotype and overlay methods are applied. The utilized modeling technique also sets scope and accuracy of the learner model. Generally, the motivation is to build an extensive learner model with as much accuracy as possible.

After the construction, the model has to be initialized. This step requires information, which allows to fill the learner model but spares the effort of the learner. Therefore, the gathered information must be used in the best possible way to infer much of information. Inferring data requires the usage of intelligent methods like Bayesian or machine learning methods which leads again into much effort for the initialization. Keeping the model of the learner up-to-date requires an observation of the learner during instruction. The only source for keeping the topicality of information is to consider the interactions between learner and system during instruction.

3.6 Summary

An adaptive system needs information about the target to which it adapts. Since this adaptation target is mostly the user of the adaptive system, a model of the user is needed.

By utilizing a user model, an adaptive system can use this user model in three different types of system actions. The system can interpret user actions in different ways concerning stored characteristics in the user model. For example, users with dyslexia can have particular problem with some words. The system recognizes these mistyping and corrects the input automatically. In the other direction, the machine actions or the output of the system can be tailored to fit the needs of the user, and finally, internal actions can be influenced by the information stored in the user model.

In the context of adaptive e-learning systems, a user model stores information necessary to adapt the instruction. Macro-adaptive systems employ a simple model of the user. The instruction is adapted to the user beforehand and no adaptation is processed during instruction. User models used in ITS and AEHS are more sophisticated and store more information about the user. The focus lies on the interactions between user and system during instruction and on the domain knowledge. This allows to adapt the system to the preferences and the current knowledge of the user.

Considering the application of a user model in an adaptive e-learning system, the user model is represented by a learner model. Thus, the difference between a

user and a learner model is the specific utilization of the learner model. A learner model is mainly applied in e-learning systems, while a user model is more generally and does not focus on a specific application domain. The content of a learner model and how this content is arranged was described in Section 3.4.

A learner model has to be constructed, initialized and updated. These processes are described in Section 3.5 as modeling techniques. The initialization of a learner model is an important topic, where an appropriate way to gather the required information has to be found. Especially, the effort for the user during the initialization must be considered since this process affects the accuracy and the usability of the learner model and of the overall system. To keep the stored information about the learner up-to-date, changing information about the learner must be included in the learner model. After the information within a learner model is changed the new information must be delivered. The delivery affects systems, which use the learner model and should keep the information consistent over all places where it is used.

The information stored in a learner model varies between different models and often depends on the surrounding or using adaptive e-learning system. To provide as much interoperability for a learner modeling system to be used by several systems, it is necessary to agree on contained information in a learner model. This agreements are represented by standards. There exist several standards in the field of user modeling, which are described in the following Chapter 4.

4. Standards for User Modeling and Profiling

4.1 Introduction

Currently numerous organizations, consortia, etc., like e.g. the Dublin Core (DC) Metadata Initiative, the Institute of Electrical and Electronics Engineers (IEEE), the Instructional Management System Global Learning Consortium (IMS GLC) the Advanced Distributed Learning Initiative (ADL), are working in the area of e-Learning standards [Paramythis and Loidl-Reisinger 2003]. Though, only some of them are dealing with the profiling and modeling of the user.

In this chapter, some standards in the field of user profiling and modeling are discussed. Section 4.2 gives a glimpse to the "low-level" standards vCard, eduPerson and ULF, where only basic information of the user is stored. The more important standards that have been analyzed and are referred to in the subsequent sections are GESTALT[1], PAPI Learner[2] (henceforth referred to simply as PAPI) and IMS LIP[3] (see Sections 4.3 - 4.5).

This chapter gives a raw overview of what is currently available and what is covered within these standards. The goal is to emphasize the advantages and disadvantages of each standard by comparing them against each other.

4.2 Basic Standards

This section depicts three basic standards, vCard, eduPerson and ULF. These standards are applied in the field of e-learning but do not represent a complete user modeling standard since important facets of the user which are relevant for adaptive systems are missing.

4.2.1 vCard

The concept of vCard[4] relies on representing the kind of personal and business information which is usually covered by a business card. Such information is for example name, address, date of birth, e-mail address, etc. [IMC 1996]

Since no accurate information needed for personalization in e-learning is stored within the vCard standard, it lacks of the capability of being used for tailoring the learning content to the customer's needs. Therefore it is not applicable in the context of e-learning directly but it can be seen as a standard used as a basis for user profiles. [Stratakis *et al.* 2003]

[1] `http://www.fdgroup.com/gestalt/`
[2] `http://edutool.com/papi/`
[3] `http://www.imsproject.org/profiles/`
[4] `http://www.imc.org/pdi/`

4.2.2 eduPerson

The eduPerson[5] standard is a schema to enable the transfer of information about people involved in higher education. According to [Stratakis *et al.* 2003], eduPerson is used by US universities. It covers the participants student and academic staff. Covering both roles has the advantage of being able to deal with all parties participating in a learning and teaching process.

Comparing eduPerson to vCard [Stratakis *et al.* 2003], eduPerson concede some additional specified characteristics such as affiliation, description, entitlement and preferred language which extends the vCard standard. Nevertheless, this extension is only a small step forward in the direction of a usable standard in the context of e-learning.

4.2.3 Universal Learning Format (ULF)

ULF [ULF 2000] was developed by Saba Software and is a standard based on DC, vCard and other educational metadata standards [Stratakis *et al.* 2003].

The ULF standard describes not only the information about the learner, but also the learning content defining formats for catalogs, learning contents, competencies, profiles and certifications. According to [Stratakis *et al.* 2003], ULF uses the Resource Description Framework (RDF) for the resource description and the discovery.

As mentioned above, ULF covers both, the description of the learner information and the learning content. This has the advantage that major parts of an e-learning system can be implemented following the same standard. Concerning the implementation and realization of the ULF standard the proprietary rights might be an important drawback since ULF is the product of a corporation, namely Saba Software Inc. which holds the rights for this standard.

4.3 GESTALT

Getting Educational Systems Talking across Leading Edge Technologies (GESTALT[6]) is a project financed by the Advanced Communications Technology and Services (ACTS[7]) program [Conlan *et al.* 2002a]. The goal of the ACTS program is to

> *"accelerate deployment of advanced communications infrastructures and services, and is completed by extensive European research in the related fields of information technology and telematics."* [ACTS 2005]

Therefore, a lot of projects were supported by ACTS which can be found on the program web site.

[5] eduPerson is specified in [eduPerson 2004]

[6] `http://www.fdgroup.co.uk/gestalt/`

[7] `http://www.cordis.lu/infowin/acts/analysys/intro/index.html`

[Wade *et al.* 2002] describes the aim of the GESTALT project as a development of a web-based learning environment being able to provide a variety of learning materials and resources.

The basic process of creating and storing a user model used by GESTALT is described in [Conlan *et al.* 2002a]. Starting with the construction of a user profile, GESTALT gathers the required information by means of user questionaries. Theses questionaries are arranged and handled by a wizard which guides the user through a sequence of question forms.

[Conlan *et al.* 2002a] lists the obtained information in form of personal details, contact details, qualification details, skill details, learning preferences and mode of delivery. After the user profile is constructed, a user model is established using the previously gathered profile information. This profile is then stored locally (on the machine of the user) in form of an XML document.

This section examines the GESTALT project by starting with the architecture followed by the used data model (see following Sub-sections 4.3.1 and 4.3.2).

4.3.1 Architecture

The detailed architecture of the GESTALT project is described in [Wade *et al.* 2002] which served also as reference for this sub-section. Thus, a raw overview over the main issues is given in the following.

The purpose of the GESTALT project is described by Wade *et al.* as follows:

> *"The objective of the GESTALT architecture is to establish a framework for the development of compatible, heterogeneous, scalable, and distributed educational systems."* [Wade *et al.* 2002]

The GESTALT system allows persons to find out if the needed resources for their education exist and if they exist, to request these resources to be transfered to their local learning environment. The delivery of the requested resources is carried out by an established infrastructure.

The GESTALT architecture is presented in form of its Functional Architecture (see Figure 4.1), which defines the functional components of the overall system and has been derived from the educational actions, roles, and relationships defined in the GESTALT business model and the requirements specification presented in the GESTALT deliverable D201[8].

[8]Document available on: `http://www.fdgroup.co.uk/gestalt/d201v2.zip`

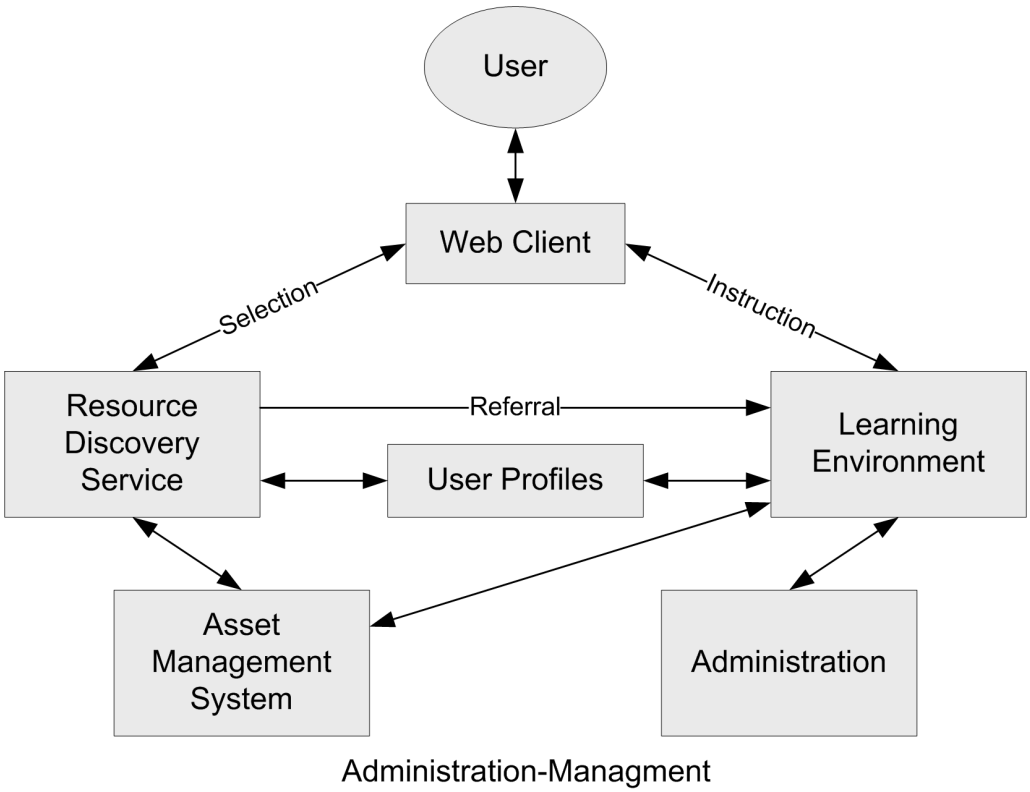

Figure 4.1: GESTALT Functional Architecture [Wade *et al.* 2002]

As shown the Figure 4.1, the complete functional system consists of several functional components. The six main components are:

- Web Client
- Resource Discovery Services
- Learning Environment
- User Profiles
- Asset Management System
- Administration

GESTALT assumes that all services available for the end user are delivered to the local machine by using a small and light weight client technology. Therefore, the *Web Client* is running on the user machine and has access to the *Resource Discovery Services* for the *Selection* action and to the *Learning Environment* for the *Instruction* action.

The *Resource Discovery Service* provides the user with abilities to explore which courses are available and which institutions are offering these courses. The tasks of the *Learning Environment* are to provide complete online support for the process

of learning, to track the progress and the achievement of the learner, to support a flexible and modular curriculum and the organizational management of the learner. The *Asset Management System* controlls the access to resources. Some resources might be for public usage while others are limited to learners, which have subscribed for this specific resources. The *Administration* component provides administration abilities for the institutions, which offer courses.

The *User Profiles* is a directory service used to store user preferences for the *Learning Environment* interactions, such as configurations, and is the central component in this architecture. Additional settings for the *Resource Discovery Service*, like for example often searched hosts, can also be stored by the *User Profiles* component.

4.3.2 Data Model

The applied data model in the GESTALT project is described in detail by [Wade *et al.* 2002] which is also the underlying reference for the following sub-section. This sub-section gives a brief overview over the data model by summarizing the main issues.

The data model used within the GESTALT project is based on the following three standards:

- Unit Object Model (UOM)
- Learning Object Model (LOM)
- Public and Private Information (PAPI)

The *UOM* is used to represent a learning experience. A learning experience is for example, a degree course. Therefore it has to include learning goals, required outcomes, the path through which a learner may run and the specific resources which the learner may use to learn. The applied *LOM* standard covers the learning resources and learning materials (for example books or lectures). For the learner's information and the learner's progress through the learning experience, the data model applies the *PAPI* standard (see following Section 4.4 for details).

Based on these three standards, GESTALT has adopted them and developed a new standard called Extended PAPI (EPAPI). According to [Rodríguez-Estévez *et al.* 2003], the EPAPI standard is an XML implementation of PAPI but tailored to the requirements of the GESTALT project.

The GESTALT project represents a complete learning environment, including a defined architecture and a data model. GESTALT applies three accepted standards to handle the required data. The user model applied by the GESTALT project is based on the PAPI standard which is depicted in the following section.

4.4 Public and Private Information - PAPI Learner

The Public and Private Information for Learners (PAPI Learner) standard takes ideas from Intelligent Tutoring Systems (ITS) and incorporates relationships between persons. Since performance information is the most important part of ITS [Dolog and Nejdl 2003], it plays also a major role in the PAPI Learner standard. Compared to the already described standards in Section 4.2, the PAPI Learner standard specifies the model of a learner with all its forms, such as syntax and content. [Russell 2003] describes PAPI Learner with the following sentence:

> *"PAPI (Personal and Private Information) specifies the syntax and semantics of a 'Learner Model', which will characterise a learner (student or knowledge worker) and his or her knowledge/abilities."*

Thus, PAPI Learner has to include many more parameters than former shown standards. [Russell 2003] lists parameters like knowledge, skills, abilities, learning styles, records and personal information which are covered within PAPI Learner. But not only the parameters are important, also the depth of detail must be considered. PAPI Learner allows parameters to be presented with an adjustable focus, starting with an overview down to the smallest units.

As mentioned in Sub-section 4.2.2 eduPerson deals with different roles or users of the system, and so does PAPI Learner. According to [Russell 2003], PAPI Learner enables different points of view, which are for example learner, teacher, parent, school, employer and so on.

[Russell 2003] identifies a key feature of the PAPI Learner standard in its logical division. For example, the security and the administration of the learner information is separated. Sub-section 4.4.1 describes the overall structure of PAPI Learner. Further, the learner information is separated into six groups, called learner information groups (see Sub-section 4.4.2).

As indicated by its name, PAPI Learner emphasizes the importance of privacy and security (see Sub-section 4.4.3) for a learner model [Russell 2003]. Finally, a summarization of the PAPI Learner standard is depicted in Sub-section 4.4.4.

4.4.1 Common Features, Information Types and Bindings

In general, the structure of PAPI Learner is divided into three main areas, which are common features, information types and bindings. These areas of the PAPI Learner standard are described in detail by [Farance 2001], which is also the foundation of this section.

The three main areas of PAPI Learner are again separated into smaller Parts (see Figure 4.2) where each Part represents a logical unit of the area. Various implementations of PAPI Learner may cover only some parts of the standard but are still conform to the standard.

The area of *Common Features* covers the general features which have to be implemented, such as the *Core Features* (Part 1), the *Data Element Registry* (Part 6)

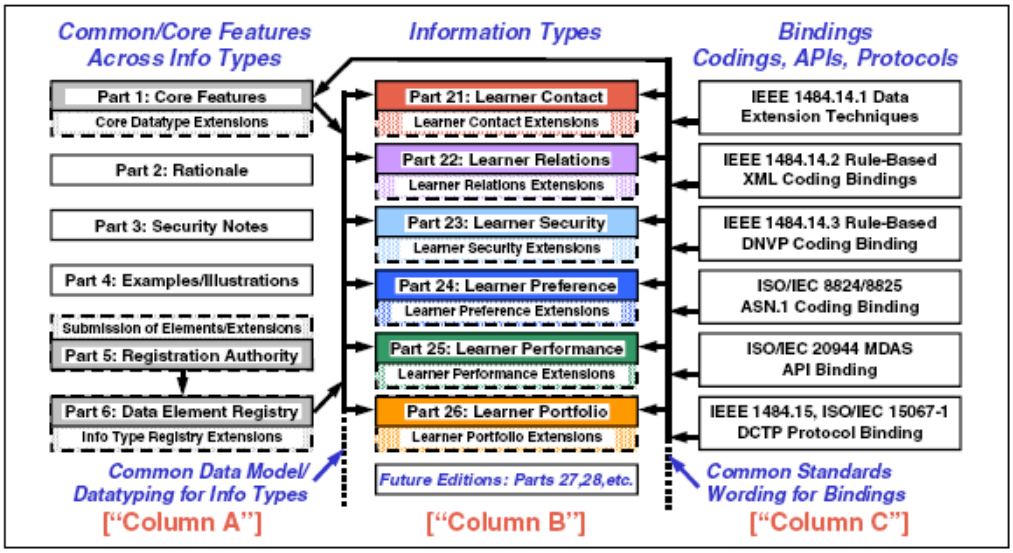

Figure 4.2: Relationship among the Parts of the PAPI Learner Standard and relationship to other standards. [Farance 2001]

and the *Registration Authority* (Part 5) (see Figure 4.2), and are available among all realizations of PAPI Learner. An implementation does not necessary include the Parts 2-4 to claim conformity to the PAPI Learner standard. The *Core Features* include datatypes that are used by other parts of PAPI Learner. Information about data elements, such as enumerated value spaces, are covered by the *Data Element Registry*. *Registration Authority Process* gives a description of how to maintain the *Data Element Registry*. The Parts *Rationale*, *Security Notes* and *Examples/Illustrations* are only instructive guides and hence need not to be considered by the implementation.

The next area covers the *Information Types*. In general the *Information Types* (Parts 21-25) can be considered as learner information types as described in Subsection 4.4.2. To be flexible for further versions of the PAPI Learner standard, the Parts 27-28 are included.

The last area is called *Bindings* and provides a mapping to various standards, specifications and technical reports.

To describe a specific implementation of PAPI Learner, the structure of Figure 4.2 is often used by referring to the three shown columns (Columns A-C) and corresponding Parts. For example a specific implementation agrees to the Parts 1 and 6 out of Column A and Part 21 from Column B, using an XML binding (Column C). This gives the opportunity to characterize the implementation in a simple and clear way.

4.4.2 PAPI Learner Information Groups

The overall structure of the PAPI Learner standard was already discussed in the previous section. The current section focuses on the six groups of learner information in the PAPI learner model (see Figure 4.3), which are specified in [Farance 2001], and describes the information types of the PAPI Learner standard as well. This section is based on the specification of the PAPI Learner standard, given by [Farance 2001].

These six Learner Information Groups cover, as its name already implies, the information about the learner in all details. The Learner Information Groups can be considered as the core and the purpose of the PAPI Learner standard.

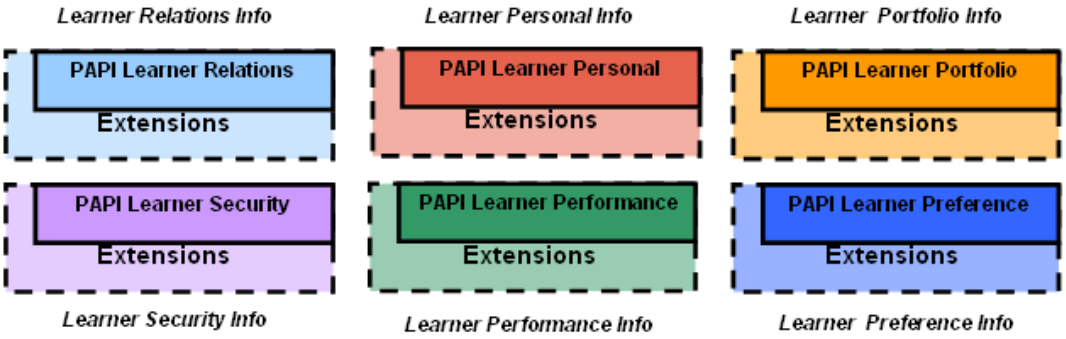

Figure 4.3: PAPI Learner Information Groups. [Farance 2001]

The Learner Information Groups divide the available information about the learner into six independent logical units, namely:

- Performance Information
- Preference Information
- Portfolio Information
- Relations Information
- Security Information
- Contact Information

The PAPI Learner approach separates the learner information according to these six units. The utilizers of the PAPI Learner standard are not restricted to keep this structure. Thus, they are free to reassemble parts of the learner information to fit their needs. In the following, a short overview over these units is given.

Performance Information

The *Performance Information* contains the history, the current work and the future goals of the learner. This unit is created and used by the components of the learning environment. Further, it can be used by the learning environment to improve or optimize the learning experiences for the learner.

Preference Information

The *Preference Information* unit describes preferences that may improve human-computer interactions for the learner, such as for example the preferred input and output devices.

Portfolio Information

The *Portfolio Information* represents the work of the learner as a collection of actions. Further, it keeps references to the work done by the learner. The *Portfolio Information* is designed to exemplify and to assess the achievements and the skills of the learner, such as passed exams, written articles and so on.

Relations Information

The *Relations Information* stands for the relations of the learner to other users of the learning environment. Such users are for example teachers, supervisors and other learners.

Security Information

The *Security Information* holds the security credentials of the learner. Such credentials are for example passwords, private and public keys but also biometric data.

Contact Information

The *Contact Information* is used by the administration and includes for example name, address, place of birth and so on. To keep such information away from public access, the *Contact Information* is typically hold private and secure.

4.4.3 Public and Private Information

As already depicted in Sub-section 4.4.1, users of PAPI Learner are not forced to use the proposed structure of the learning information. PAPI Learner allows to combine different parts to fit the needs of the user.

The combination of parts of the learning information may reveal privacy and security problems when sharing the information with other organizations. For example, combining the contact and the performance information might be useful for the current application, but privacy violations occur when this information is shared. In this case there are conflicts between the privacy level of the contact information and the performance information. Such privacy issues were the main reason why PAPI Learner split the learner information into six units. However, PAPI Learner does not specify which unit of the learner information is private and which is public.

PAPI Learner permits to handle the units of the learner information in different ways with respect to privacy. For example the learner contact information is private and secure while the learner preference information is marked as public. The level

of security and privacy is selected by the administrator. An administrator can be the learner for its own learner information, or the administrator of the organization.

4.4.4 Summary

As discussed in this section, PAPI Learner is a complete standard regarding storage of user information for learning environments. Nevertheless, PAPI Learner is easy to extend and is adaptable to fit new requirements for different applications. Such applications are for example medical and financial applications. [Russell 2003]

Further, PAPI Learner is not limited onto one role within the system, but it is able to model different points of views, such as for learner, teacher, staff and employer.

Compared to previously discussed standards, PAPI Learner also focus on the topic of privacy and security, which is a central point of interest when developing a user modeling system.

Beside GESTALT and PAPI Learner exists a third important standard specification called IMS Learner Information Package, which is depicted in the following section.

4.5 IMS Learner Information Package

The IMS[9] Learner Information Package (LIP) is a specification for a standard to record information about learners [Wilson and Jones 2002]. Version 1.0 was released in March 2001 and the current version is 1.0.1 with some minor changes to the original version 1.0 [Smythe *et al.* 2001].

The underlying reference for the following section is the specification of the LIP standard [Smythe *et al.* 2001], which includes the complete information about LIP.

LIP is designed to hold information about the learner, including his progress and received awards. Further, LIP enables the transfer of this information between different software applications.

To be more precisely, LIP is a collection of information about a learner or a producer of learning content. The roles are not limited to one single learner, groups of learners can be handled as well. Producers of learning content may be organizations or individuals and are separated into three divisions, namely Creators, Providers and Vendors, with different tasks and rights. To provide the ability of exchanging this information among different applications, the information is split into several packages.

The arrangement of LIP consists of a set of packages, called segments or categories. The following section deals with these segments and gives an overview over the structure of LIP. Sub-section 4.5.2 shows how the structure is built in form of an XML schema. Privacy and security mechanisms offered by LIP are examined in Sub-section 4.5.4 and finally, some implementations of the IMS LIP standard are introduced in Sub-section 4.5.4.

[9] `http://www.imsproject.org/`

4.5.1 The Structure of IMS LIP

The structure of IMS LIP consists of segments or categories and elements. The information about the learner is split into eleven segments, while the elements specify the data and the structure of a segment.

Segments

LIP divides the learner information into eleven segments, starting from the *Identification* to more administrative content like *Securitykey* (see Figure 4.4).

The reason for this separation is described in [Smythe *et al.* 2001] as follows:

> *"These categories were chosen to meet the requirements of a large variety of use cases and to facilitate mapping among IMS and other relevant specifications."*

According to [Smythe *et al.* 2001], the learner information is divided into following segments:

- Identification
- Goal
- Qualification, Certifications and Licenses (QCL)
- Activity
- Interest
- Competency
- Accessibility
- Transcript
- Affiliation
- Securitykey
- Relationship

The *Identification* segment holds the basic information that helps to identify a person. Elements like name, address, e-mail, etc., are contained within this segment. Further, biographic and demographic data, which is relevant in the context of learning should be noted here as well.

Personal goals and ambitions are stored within the *Goal* segment. The status of each item in this segment can be tracked and an encapsulated structure allows to store sub-goals.

The *Qualification, Certifications and Licenses (QCL)* segment reflects the accomplishments already fulfilled, combined with a structure where the source of the *QCL* and its level is handled. An example would be "journeyman, plumber".

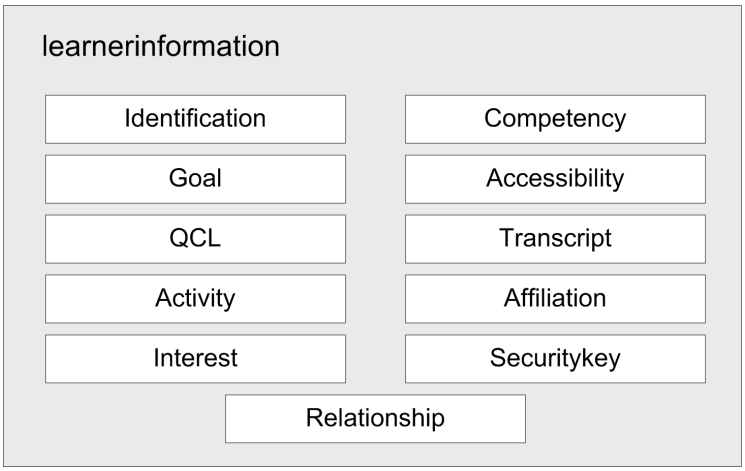

Figure 4.4: The IMS Learner Information Package (LIP) Core Segments. [Smythe *et al.* 2001]

The *Activity* segment contains training or education work of the learner. It is not limited to traditional education institutions. The *Activity* segment is kept flexible to allow the coverage of different activities. This segment does not just simply record activities and outcomes, in fact it offers the ability to record the digital representation of the activity, like e.g. a code sample or the digital representation of a painting in the field of art.

Interest represents hobbies of the learner and other leisure time activities. The content may be related to the *QCL* data and may also include digital representations of the activities.

The *Competency* segment holds information about abilities or skills the learner has gained. It is possible that these skills are connected to other information from the *Activity* and the *QCL* segment.

General access to the learner information is stored in the *Accessibility* segment. Such accessibilities are defined by language capabilities, disabilities, qualifications and learning preferences. Learning preferences include cognitive preferences such as learning styles, physical preferences (for example a preference for a large font) and technological preferences such as a preference for a particular operating system.

The *Transcript* segment is a placeholder for standards from other organizations. It enables the concept of storing records in external data formats. As an example [Smythe *et al.* 2001] mentions the US University Academic Transcript. Further, it is possible to store such transcripts as PDF documents.

Descriptions of the organizations, which are related to the learner, are stored within the *Affiliation* segment. This may include for example work groups, professional association and clubs where the learner has a membership.

Passwords and security keys are hold in the *Securitykey* segment, which is then used during the transaction of learner information. Before a transaction is processed, it is checked for granted rights stored in the *Securitykey* segment.

Since the core structures of IMS LIP do not include relations or links between

core segments themselves, the *Relationship* segment is needed to specify these connections. This leads to a simplified administration of the links, since only the *Relationship* segment is needed to manage such connections. The *Relationship* segment is used to store the description of the relations between data in other segments.

The second term used to describe the IMS LIP structure is *Element*. *Elements* are described in the following sub-section.

Elements

An element can be seen as a part of a segment and can be specified as data types (for example language strings) or as recursive hierarchical structures. Elements also support referencing mechanisms, such as internal references, external references and references described by a Universal Resource Identifier (URI).

The specification of the IMS LIP standards covers a lot of different data elements to be able to support a wide range of requirements by different learning environments. However, IMS LIP was designed to offer ways for supporting specific needs of the actual implementation. Thus, the implementation has the possibility to extend an element by its own needs.

4.5.2 XML Schema

IMS LIP uses an XML schema, rather than a Data Type Definition (DTD), as the binding. This section shows how the structure of IMS LIP is implemented in XML. It is possible to define element names within the document by using XML schemes [Smythe *et al.* 2001], but the opportunity of using other bindings is not excluded.

The description of the used XML schema by IMS LIP can be found in [Smythe *et al.* 2001]. This document is also the basement for this section.

The XML schema defines elements, the content of these elements and their attributes. Further, it defines the vocabulary used within the IMS LIP standard. A short example of how IMS LIP looks like is given in Listing 4.1.

```
<language>
  <typename>
    <tysource sourcetype="imsdefault"/>
    <tyvalue>German</tyvalue>
  </typename>
  <contentype>
    <referential>
      <indexid>language_01</indexid>
    </referential>
  </contentype>
  <proficiency profmode="OralSpeak">Excellent</proficiency>
  <proficiency profmode="OralComp">Excellent</proficiency>
  <proficiency profmode="Read">Good</proficiency>
  <proficiency profmode="Write">Poor</proficiency>
</language>
```

Listing 4.1: An example of a small portion of a LIP record being used to record preference information about the learner. In this case language proficiency.

The following section deals with the XML binding used by IMS LIP in detail.

XML Binding Description

This section describes the XML format, proposed in the IMS LIP standard by picking out some of the main elements such as `<learnerinformation>` and some of its sub-elements like `<identification>`, `<transcript>` and `<securitykey>`. This structure can be seen in form of a tree in Figure 4.5.

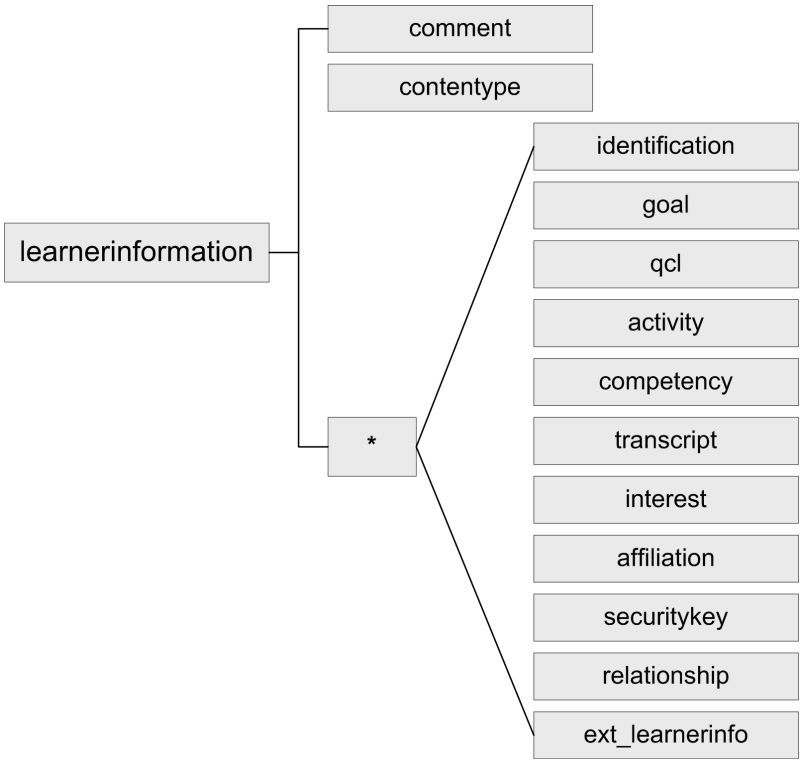

Figure 4.5: <learnerinformation> elements [Smythe *et al.* 2001]

The `<learnerinformation>` element is the outermost container for the learner information. There is only one `<learnerinformation>` element in each XML file. The information within this element is specified as the collection of segments in IMS LIP, which are already discussed above in Sub-section 4.5.1. The content of the complete `<learnerinformation>` element can be spread over several XML files to facilitate the exchange of the information.

The *learnerinformation* element contains the following elements:

- <*comment*> This element contains the comments that are relevant to the structure as a whole. It occurs not more than one time within the `<learnerinformation>` element but is not necessarily a part of it.
- <*contentype*> Contains the content meta-data description concerning the index for the data, access rights and timestamps. It occurs zero times or once within the `<learnerinformation>` element.

- <*ext_learnerinfo*> This element contains the extensions of the `<learnerinformation>` element. It occurs zero or more times within the `<learnerinformation>` element.
- The eleven Learner Information Segments (see Sub-section 4.5.1):
  - <*identification*>
  - <*goal*>
  - <*qcl*>
  - <*activity*>
  - <*competency*>
  - <*transcript*>
  - <*accessibility*>
  - <*interest*>
  - <*affiliation*>
  - <*securitykey*>
  - <*relationship*>

The `<identification>` element contains information to identify the learner. There is only a single instance of the `<identification>` element within each `<learnerinformation>` element. The types of information included are names, addresses, contact information, demographics and representative agents (see Figure 4.6).

To visualize the `<identification>` element see Listing 4.2 where an example of the `<name>` element is shown.

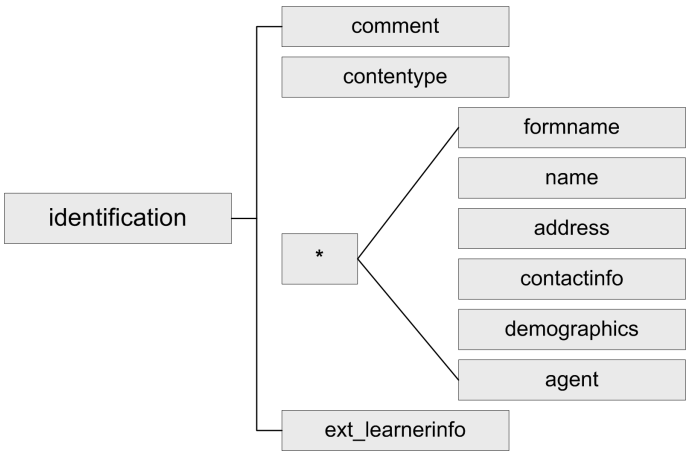

Figure 4.6: <identification> elements [Smythe *et al.* 2001]

```
<name>
  <typename>
    <tysource sourcetype="imsdefault"/>
    <tyvalue>Preferred</tyvalue>
  </typename>
  <contentype>
    <referential>
      <indexid>name_01</indexid>
    </referential>
  </contentype>
  <partname>
    <typename>
      <tysource sourcetype="imsdefault"/>
      <tyvalue>First</tyvalue>
    </typename>
    <text>Bob</text>
  </partname>
  <partname>
    <typename>
      <tysource sourcetype="imsdefault"/>
      <tyvalue>Last</tyvalue>
    </typename>
    <text>Dylan</text>
  </partname>
<name>
```

Listing 4.2: Example of the <name> element

The `<transcript>` element is used to store the summary records of the academic performance at an institution. A tree representation of the `<transcript>` element can be seen in Figure 4.7. This information may be hold in an undefined level of detail. Thus, there is no specified structure for a transcript. Each entry of a transcript records uses a separate `<transcript>` element.

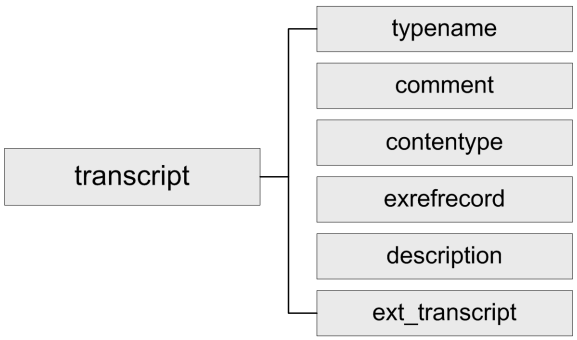

Figure 4.7: <transcript> elements [Smythe *et al.* 2001]

At last, the `<securitykey>` (see Figure 4.8) is described. This segment defines and controls the levels of privacy and security for a specific learner. Each key or entry is stored in a separate `<securitykey>` element.

4.5.3 Data Protection

As already mentioned in Sub-section 4.5.1, IMS LIP enables the inclusion of mechanisms for maintaining privacy and security by providing the *Securitykey* segment.

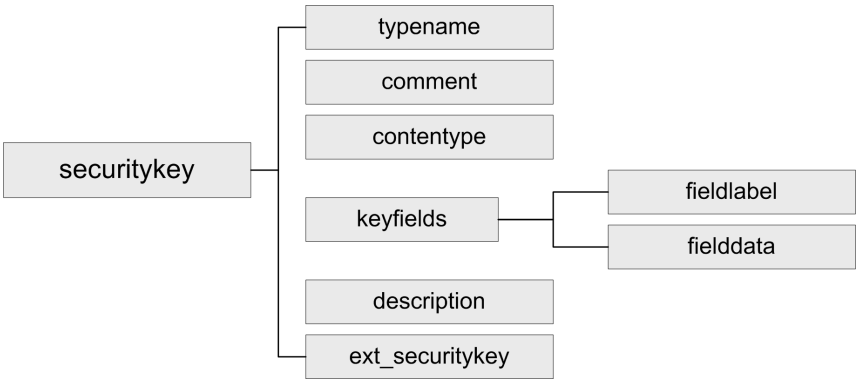

Figure 4.8: <securitykey> elements [Smythe *et al.* 2001]

However, the specification of the IMS LIP standard does not provide any solution or restrictions how privacy and security must or should be handled [Russell 2003]. This has to be defined by the implementation, with respect to the specific needs and requirements.

4.5.4 Implementations

The website of IMS[10] lists organizations and products, which are using IMS specifications. There are links to the organizations and products, including short overviews of the implemented and used standards, name of the project/organization, etc.. Currently IMS indexes four organizations and two projects, which use or support the IMS LIP standard.

Additionally, [Russell 2003] refers to two projects that use IMS LIP. The first project is called Southwest Hosts Enhancing Lifelong Learning (SHELL[11]). The second project is Northern Ireland Integrated MLE (NIIMLE[12]). NIIMLE was launched in February 2003. According to [Russell 2003], both projects use IMS LIP to offer students the possibility to take courses and modules from any partner institutions. After finishing the taken course or module, the credits are added to the students transcript.

Further, the Joint Information Systems Committee (JISC[13]) decided that every participating organization needs to support IMS standards [Olivier 2002] which implies the usage of the IMS LIP standard [Russell 2003].

4.5.5 Summary

Summarizing the information from this section, it can be concluded that IMS LIP produces a complete representation of learner information for e-learning systems.

[10] http://www.imsglobal.org/direct/directory.html
[11] http://www.educationaldevelopment.net/shellproject
[12] http://www.niimle.ac.uk/
[13] http://www.jisc.ac.uk/

Considering the possible binding in form of an XML-schema, IMS LIP provides abilities to be used by applications in the context of personalization and in every utilization where structured information is needed.

As depicted in Sub-section 4.5.4, there exist systems which implement or provide the IMS LIP standard.

For the utilization of IMS LIP in the context of e-learning, [Paramythis and Loidl-Reisinger 2003] reports that IMS LIP lacks in some way of covering user actions and interactions with the system. Thus, it can be concluded that IMS LIP alone finds only small usage in adaptive e-learning where the actions and the skills of the user regarding the environment are important.

4.6 Summary and Conclusion

The shown standards for learner modeling in this chapter (GESTALT, PAPI Learner and IMS LIP) model the user from a rather rough point of view. But all of these three standards provide a progression of the learner model over time. The IMS LIP standard (see Section 4.5) for example, combines the results of educational activities in an overview, and provides rather static information about the learner, such as demographic information.

The presented standards are only of limited use in the context of adaptive e-learning since the coarse grained level of detail excludes the needed detailed information. [Paramythis and Loidl-Reisinger 2003]

According to [Rousseau *et al.* 2004], adaptive e-learning systems require a history of the user's interactions. The system must be able to adapt the content to the need of the learner. Concerning the necessity in adaptive e-learning of basing adaption processes onto the knowledge and skills of the learner, a sort of relation, which represents the status of the learner regarding learning units, is needed.

Following [Paramythis and Loidl-Reisinger 2003], PAPI Learner (discussed in Section 4.4) can be seen as the only standard that fulfills the requirement of detailed information about the activities of the learner. PAPI Learner has advantages with respect to this issue because it includes ideas from ITS, where the learner performance is seen as the most important information, as mentioned in [Dolog and Nejdl 2003].

Although PAPI Learner has its advantages in the context of adaptive e-learning, it lacks in the topicality and further development of its specification. Since IMS LIP is based on PAPI Learner, it might be a good reason to take IMS LIP as the standard when developing a learner modeling system. IMS LIP has its advantages in its topicality (the current version was released in January 2005), in providing an extensible structure and in its usability.

After acquiring knowledge about the basic concepts about adaptive e-learning in Chapter 2, the different aspects of user models and the available modeling techniques from Chapter 3, and the standards in user modeling described in this section, it is now possible to take a closer look at the available user modeling systems, which are introduced in the following Chapter 5.

5. State of the Art of User Modeling Systems

5.1 Introduction

Although the topic of user modeling is not new, the term user modeling itself has become public in 1980. Previously, characteristics about the user were utilized to change system behaviors and properties. Early deployment of user characteristics were made in the fields of information retrieval and dialog systems. Further, there was no clear distinction between user characteristics components and other system components. This period is described in this chapter as early user modeling (see Section 5.2).

The first user modeling system is the "General User Modeling System" (GUMS), published by Tim Finin in 1986 (see Sub-section 5.2.1). GUMS is not connected to any specific purpose or part of any system. This can be seen as the basis for further work and leaded - among other solution approaches - into user modeling shell systems (see Section 5.3). Shell systems are modeling systems without any specified purpose or application. Thus, they are manifold applicable compared to the early information retrieval or dialog systems, where the modeled user characteristics are strongly connected to the application itself.

The development of the World Wide Web (WWW) entailed the development of server systems. To satisfy the need of this development for having access to the user model at every location, user modeling systems were transfered onto servers. User modeling servers, which are described in Section 5.4, provide the possibility to store user models centralized and providing access within the connected network.

The depicted user modeling shell systems, user modeling servers and examples which represent current efforts, form the current state of the art in the field of user modeling for this thesis.

5.2 Early User Modeling Systems

Early attempts to user modeling have been made in the fields of dialog systems and human-computer-interaction. As an example of a dialog system, which utilizes a user model, GRUNDY ([Rich 1979]) is described.

GRUNDY calculates recommendation of books according to its assumptions about the user's personal characteristics. Such characteristics include for example, educational and intellectual level, preference for thrill, fast-moving plots or romance, tolerance for descriptions of sexuality, violence and suffering. All these characteristics are represented by values within a linear scale including associated certainty ratings. The initialization of the user model is processed by using stereotype methods (see Sub-section 3.5.1) based on given answers to questions during the first usage of the system. For example, from the fact that the user has a male first name, GRUNDY infers a high sex tolerance and a low one for romance.

In the early 1980's, user modeling for user interfaces became popular and is applied in form of adaptive user interfaces. The applied user models were very small. Mainly the user models recorded command usage or data access and provided an automatic response to the frequency of this usage. For example, the inference was that the more a user uses a certain command, the more it would be likely, that this user is going to use this command in future. Thus, this command was placed higher in the command list. There was no attempt to infer or represent any other information about the user, nor to maintain a long-term representation of user characteristics. Such user models are necessarily limited because of their localization to a specific application. [Murray 1987]

The early approaches of user modeling are application-oriented since they store only relevant information for one application. Further, these user models are not separated from other parts of the system. There is no explicit functional component responsible for gathering and storing information about the user. This step in development, towards a generic user model system is seen as the beginning of user model systems, which are not related or involved to any application. The following section describes the first separate user modeling system.

5.2.1 GUMS

In the year 1986, the "General User Modeling System" (GUMS) [Finin and Drager 1986] was published. GUMS was the first system, which allowed to abstract from the application system. The term "general" in the name GUMS describes the attribute of application-independency, which was one motivation for Finin and Drager to publish their modeling system.

The GUMS system allows to define simple stereotype hierarchies in form of a tree structure. For each stereotype it is possible to describe facts and rules prescribing the system's reasoning about it. The initial stereotype for a user must be assigned by the application. The rules can be used to derive new information, both definite and assumed, from the current information about the user. If one fact of an assigned stereotype is in conflict with an assumption about the user, this assigned stereotype will be replaced by the next higher one in the hierarchy, which does not include the troublesome fact.

GUMS supports two types of inference-rules. The *certain flag* describes information, which is definitely true and the *default flag* describes information, which is assigned to a user model by default, such as through stereotyping. For example, if a user is categorized under the stereotype Programmer, which implies that this user should know what a file is, but there is no explicit data about the user's knowledge about files, then the user knows what a file is by using the *default flag*. This information is as long valid until a contrary information will appear. On the other hand, if the user is queried about his knowledge about files, no default information is needed and the assigned property is marked with *certain flag*.

Further, GUMS accepts and stores new facts about the user, which are provided by the application system (see Figure 5.1). Further, GUMS verifies the consistency of new data with the currently stored data. If an inconsistency is found, the application system will be informed by a response to the new information action. At runtime, GUMS answers questions of the application concerning the currently held assumptions about the user.

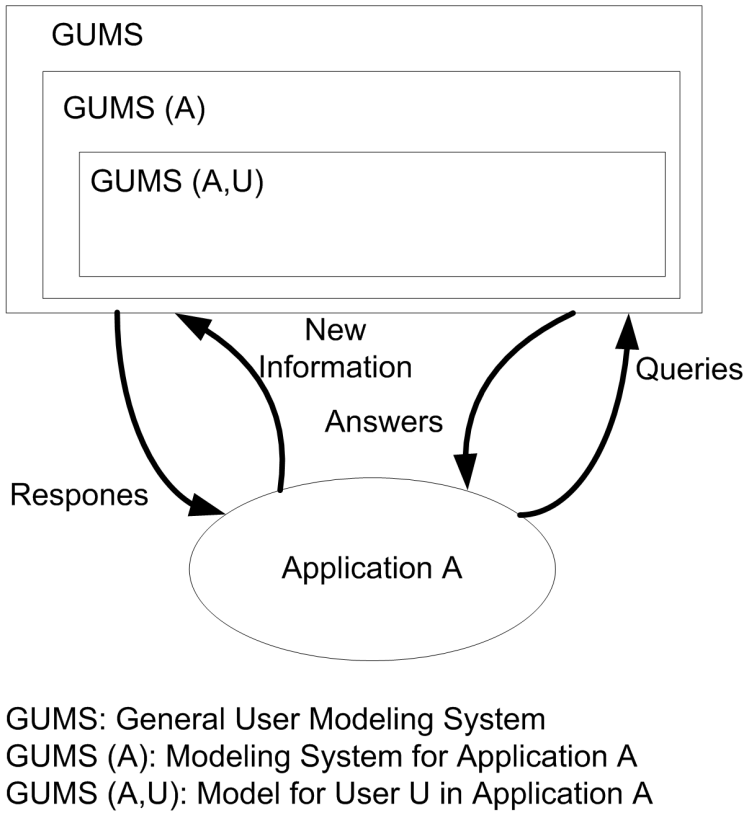

Figure 5.1: Architecture of a General User Modeling System [Finin and Drager 1986]

[Finin and Drager 1986] presented a simple architecture for a general user modeling utility which is based on the ideas of a default logic. Although GUMS was never used together with an application system, it made the basis for later user modeling systems. [Kobsa 2001a]

This section describes some aspects of the history of user modeling systems. Basically, until the publication of GUMS there was no explicit user modeling component. The required information was stored and spread over several system components. Thus, the user model was only applicable for this specific system. Since building a user model takes a lot of sophistication and efforts, general user modeling system or so-called user modeling shell systems were introduced.

5.3 User Modeling Shell Systems

User modeling shell systems provide integrated techniques, which are often used by user modeling components. The developer of a user model should be able to arrange an appropriate shell system by choosing the necessary components for the desired user modeling system. The needed components depend on the application and are filled with information about the user considering the application domain. The resulting system will fulfill all requirements for a centralized user modeling system. Information needed to initialize a user model must still be supplied by the application system.

In the following sub-sections several extensive user modeling shell systems, namely UMT, **um**, PROTUM and BGP-MS are described (see Sub-section 5.3.1 - 5.3.4).

5.3.1 UMT

The User Modeling Tool (UMT) proposed by [Brajnik and Tasso 1994] allows the specification of stereotypes, which contains descriptions of characteristics of user groups in form of attribute-value pairs. The stereotypes can be arranged in arbitrary hierarchies whereby inheritance of information to sub-stereotypes is supported. Every stereotype owns activation conditions (triggers), which define when a stereotype is applicable to the current user. For detailed information on stereotype methods see Sub-section 3.5.1.

UMT also provides a rule interpreter that allows to define inferring rules for user models. Possible contradictions between assumed user characteristics must be specified explicitly by using these rules. UMT accepts and stores assertions about a user, which are generated or gathered by the application system. Depending on the grade of reliability of these assertions it is possible to consider them as constant or as assumptions, which can be deleted later. Stereotypes with fulfilled activation conditions through preceding assumption add further assumptions, namely attribute-value pairs that characterize corresponding user groups. Some of these assumptions can be contradictory. The UMT applies after each change of the user model all inferring rules (including rules to detect contradictions) to the set of constants and assumptions. Further, dependencies between inferred assumptions are recorded.

A truth-maintenance-component determines all possible user models. That includes all consistent sets of assertions, consisting of constants, a selection of assumptions and all derivation based on the constants and assumptions. The current user model is selected by comparing the gained possible user models with given preferences. Thereby, assumptions from the application are rated higher than assumptions inferred by stereotypes. If inconsistence with new information occurs later, the assumptions involved in this inconsistence are easy to detect, since the dependencies are recorded. The set of possible user models can be revised and reevaluated to find the current user model.

5.3.2 PROTUM

The "PROlog based Tool for User Modeling" (PROTUM) was published in [Vergara 1994] and combines advantages of GUMS and UMT (see Sub-section 5.2.1 and Sub-section 5.3.1). As the name expresses, PROTUM is based on Prolog, which is a logical programming language. PROTUM contains a dependency management and a truth-maintenance system like in UMT. Stereotypes are used to infer information. The hierarchy of the stereotypes is not limited to a tree structure and the managed assumptions by truth-maintenance system are not based on attribute-value pairs like in UMT.

PROTUM calculates for each stereotype the activation rate of its triggers and uses this measure for the activation and the deactivation of stereotypes. Further, the activation rate of triggers is used to resolve conflicts between inconsistent assumptions of two activated stereotypes. For example, if stereotype A has an activation rate of 80% and stereotype B 93%, then the assumption of stereotype B is used to infer a particular information in case of a contradiction.

5.3.3 um

um [Kay 1995] is a toolkit for user modeling that represents assumptions about the user's knowledge, beliefs, preferences, and other user characteristics in attribute-value pairs. From the application system's point of view, **um** represents a library of user modeling functions. Thus, it is not an independent user modeling system in a strict sense.

Every piece of information is formed into components. A component implies the information accompanied by a list of evidences for its truth and its falsehood. Further, the source of each component, its type and a time stamp is recorded. There are 5 different types of components, *observation*, *stereotype activation*, *rule invocation*, *user input* and *told to the user*. [Kobsa 2001a]

um addresses the needs of different consumers or applications of the modeling system by assigning each **um**-consumer an individual representation of the user information. Each **um**-consumer can define a selection of tools which match its needs. This allows to create application-related user modeling systems, while the stored user information within the *um* modeling shell system is the same for all **um**-consumers. [Kay 1995]

The techniques used by the **um** system include stereotyping and rule based methods (see Sub-section 3.5.1 and Sub-section 3.5.1).

5.3.4 BGP-MS

BGP-MS[1] is a user modeling shell system focusing on modeling the user's knowledge, beliefs and goals. [Kobsa and Pohl 1995]

The user modeling shell system receives observation information about a user from the application system, processes internal classification and calculation op-

[1] BGP-MS stands for **B**elief, **G**oal and **P**lan **M**aintenance **S**ystem.

erations based on these observations. Questions from the application system are answered by assumptions about the user (system beliefs), for example about his knowledge (user beliefs) or his goals and plans.

BGP-MS works logic oriented like GUMS and UMT (see Sub-section 5.2.1 and Sub-section 5.3.1). For the acquisition of information for the user model, stereotype methods, natural language dialogs and questionnaires are used. These techniques represent the base for later inferring processes. BGP-MS consists of several components with different tasks. For example, there exists a specific component for managing stereotypes, which is responsible for the activation and deactivation of assigned stereotypes of an individual user model. Further, BGP-MS provides an integrated suite of knowledge representation mechanisms, which is based on the SB-ONE[2] knowledge-representation tool, for representing its assumptions about the current user, its domain-specific user modeling knowledge and optionally its general knowledge about the application domain.

As can be see in Figure 5.2 there are four main components in the shell system which communicate over the functional interface. The individual user model and the stereotypes are built upon the representation system which is based on SB-ONE. The user model developer can manipulate the knowledge of the representation system through the graphical interface.

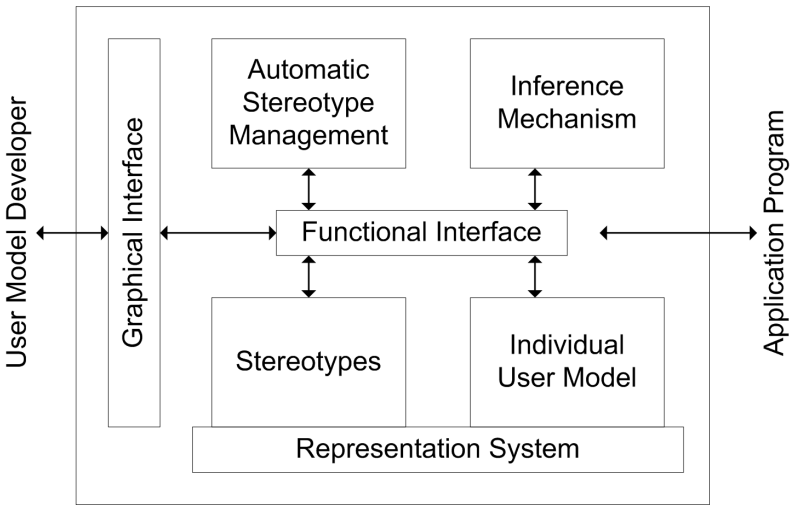

Figure 5.2: Internal view of the BGP-MS [Blank 1996]

[2]SB-ONE is a workbench for the representation of conceptual knowledge, with the emphasis on applications in natural-language systems. A good description can be found in [Kobsa 1991].

An interesting aspect of the BGP-MS is the defined communication protocol KoNstanz Inter-Process Communication Management System (KN-IPCMS). The KN-IPCMS is a platform-independent, message-oriented communication protocol which is used to communicate between an application and the BGP-MS shell system. In Figure 5.3 there are three types of communication:

1. If the application wants to tell the shell system about an observed believe or goal of a user, it can send an *bgp-ms-tell* message to the shell.

2. The message type *d-act* is used to tell the shell system about an action that a user has performed in the application.

3. Assumptions about a user are an important information for an application, so the application can request such an assumption with a *bgp-ms-ask* message sent to the shell system, and the BGP-MS can deliver the answer in a *bgp-ms-answer* message to the application.

Below the arrows in Figure 5.3, message examples are shown. These messages are in the belief, goal, plan language (BGPL). For example, the *d-act* message describes the execution of the print action on the user documentation.

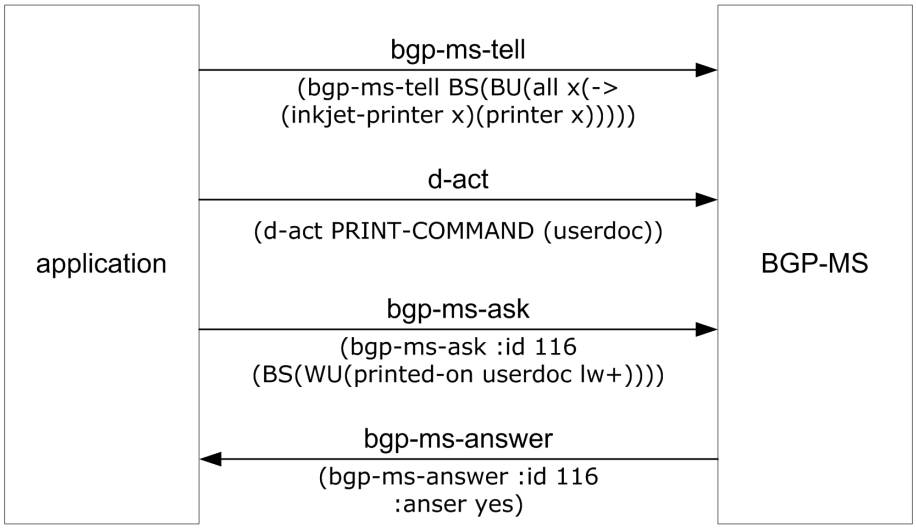

Figure 5.3: Communication between BGP-MS and the application [Kobsa and Pohl 1995]

The internal knowledge representation of BGP-MS is done by a partition mechanism. The whole knowledge is separated into several partitions, which are ordered hierarchically and allow heredity. The knowledge within partitions is described in form of predicate logic. A detailed description about the knowledge representation is beyond the scope of this work and can be found in [Kobsa and Pohl 1995]. The system can be used as a network server with multi-user and multi-application capabilities. Therefore, the BGP-MS can also be seen as a user modeling server, which are described in Section 5.4.

The presented user modeling shell systems were mainly published in the early nineties. They were not really applied and distributed, not even in the research community. One exception is the BGP-MS, which was used outside of the institution at which it was developed. However, the ideas and methods that were used in these prototypes, particular the stereotype method and the client-server architecture, made the base for later developments. [Kobsa 2001a]

The following section describes user modeling systems, which are based on a client-server architecture, where the user modeling system represents the server and the application system the client.

5.4 User Modeling Servers

The connection of computer by local area networks (LAN) and the spreading of the Internet reinforce the client-server architecture. With the gained knowledge of the user modeling shell systems, it became popular to construct user modeling systems installed on a server. This allows user modeling systems to be used by different applications in a distributed area.

The first user modeling server described in this section is the Doppelgänger system (see Sub-section 5.4.1). Doppelgänger is a system for delivering personalized news. Later, the economy picked up the idea of personalized services using the World Wide Web and user modeling servers (See Sub-sections 5.4.2 - 5.4.4).

5.4.1 Doppelgänger

Doppelgänger, published by [Orwant 1995], is a user modeling system that can monitor user actions and detect patterns within these actions. Originally, it was developed to deliver a personalized daily newspaper. The user characteristics were used to select the news.

Basically, the architecture of the Doppelgänger system consists of two levels, the sensor and the server level. The user modeling system gathers data about users from sensors, makes inferences on those data, and the results available to applications. Sensors provide data about users and may be either hardware or software. Thus, the techniques used to extract information from user activities are used within the sensors. Every sensor has its own specific purpose. For example, one sensor gathers data about the user's frequency and duration of computer use or another sensor provides data about the physical location of the user by using wall-mounted motion sensors.

To prevent or reduce influence of error prone sensors, Doppelgänger maintains an accuracy estimation for each sensor. This is used to decide how much confidence to assign to assertions based on that sensor's information. Further, the users are always able, but not forced to interact directly with the user modeling system.

The whole user model is split into several sub-models. Each sub-model covers an aspect of the user characteristics. Thus, changes in the user model are affecting only one part of the user model at one time. The information contained in a user model is encoded in a simple knowledge representation language called Sponge [Orwant

1993], which utilizes a LISP-like data structure. Listing 5.1 shows a small part of a sub-model.

```
(object orwant primary
  (object biographical_data
    (string_binding "true_name" "Jon_Orwant")
    (string_binding "e-mail_adress" "jon_orwant@uni.de")
  ...)
...)
```

Listing 5.1: A small part of a primary sub-model of Doppelgänger containing demographic data. [Orwant 1995]

Every user model is stored in a UNIX directory where a large amount of files represent *domain* sub-models and *conditional* sub-models. *Domain* sub-models contain information about a particular aspect of the user's behavior (for example, preferences for personalized newspaper content), whereas *conditional* sub-models contain information about the user that replaces *domain* information when a certain condition exists. For example, during midday a user might prefer sport news while his common interests focus on business news. In this example, the condition is a time period and becomes true during midday and overwrites the user's domain sub-model of his common news preferences.

An example for the data coming from a motion sensor is shown in Listing 5.2. This data indicates that the user *orwant* is at the place 344 complemented with time and id of the sensor.

```
((object orwant location (place 344) (time 437986473) (id active_badge))
```

Listing 5.2: Data from a motion sensor to the server. [Orwant 1995]

Before the raw data coming from the sensors is stored in the user model, it is modified by so called *learning techniques*. Doppelgänger provides three different learning techniques, namely modeling events with *linear prediction*, modeling interests with *beta distribution* and modeling location with *Markov models*. The *linear prediction* is used to predict the next occurrence of events. For example, it predicts the next time when a particular user will read the newspaper. To estimate the favor for different news topics, the *beta distribution* is used. Already gathered information about preferences for particular news topics are used to calculate the interest in a specific topic. To model the location of the user, the *Markov model* is utilized. The general purpose of the location tracking and modeling is to predict future user locations. A *Markov model* consists of a set of states, a matrix of transition probabilities and a matrix of output probabilities. A state represents the location of the user (e.g. a room, a corridor, or a desktop). Given a particular state, a Markov model describes a particular probability that the user will walk (*transition*) to another room (*state*) by the transition probabilities matrix. The matrix of output probabilities is ignored for location modeling by the Doppelgänger system. [Orwant 1995]

Another interesting aspect is the possibility to have several Doppelgänger modeling systems running in different locations. They are able to exchange their gathered information of a user or a community. Communities in Doppelgänger are similar to common stereotypes as described in Sub-section 3.5.1. If there is some information

about a user not available in his user model, default information from communities is used to supply this needed information. Some examples for communities are students, children or artists. All together there are 22 communities available in Doppelgänger. The difference between communities and stereotypes is that the membership or assignment to a community is not binary (member or no member). Each community has members with different grades of membership. A grade shows the conformity of a particular user to a community. A further difference is that a community represents the combination of its constituent user models, which means that a community represents an average of all member characteristics. [Orwant 1995]

Each user may belong to many different communities at the same time. When a default information is required for a user model **X**, each community votes an answer. The importance or strength of each vote is proportional to the similarity between the community and the user model **X**, the grade of membership. The resulting assertion is calculated by a weighted average of all the users and community models according to the similarity to the user model **X**.

5.4.2 Personis

Personis, published by [Kay *et al.* 2002], is a user model server with focus on user's privacy, control and ability to scrutinize her/his user model. A scrutable user model allows users to see the details of the information held about them and the processes used to gather it. Giving the users control over their scrutable user model allows them to make changes or enter their own estimations. Personis was mainly developed for adaptive hypertext applications but it is not limited to such applications.

The underlying user model is based on the **um** toolkit where information about a user is separated and stored in components (see Sub-section 5.3.3). The collaboration of personis with adaptive hypermedia systems (AHA 1 - AHA N) is shown in Figure 5.4. The architecture is divided into four parts, namely the server itself, generic scrutiny tools that enable the user to see and control their own user model, a collection of adaptive hypertext applications and the views, which are the conceptual, high level elements shared between the server and each application.

The generic scrutiny interface allows the user to access and modify her/his user model without the usage of an adaptive hypertext application. This scrutiny interface is application-independent. The adaptive hypertext applications are split into two parts. First, the core of the adaptive hypertext which enables the user to do some task such as 'learn to program' and second, the scrutiny interface associated with 'that' adaptive hypertext application. The user might want to scrutinize the adaptivity within the context of the adaptive hypertext application. The views of the user model available to each adaptive hypertext application define the components used by this application. For example, the AHA 1 in Figure 5.4 might need just a few components of the user model. The architecture allows the definition of a view that represents just these components. Another application could use a different view. The application developer defines the needed components of the user model required by the application and describes these components in form of a view.

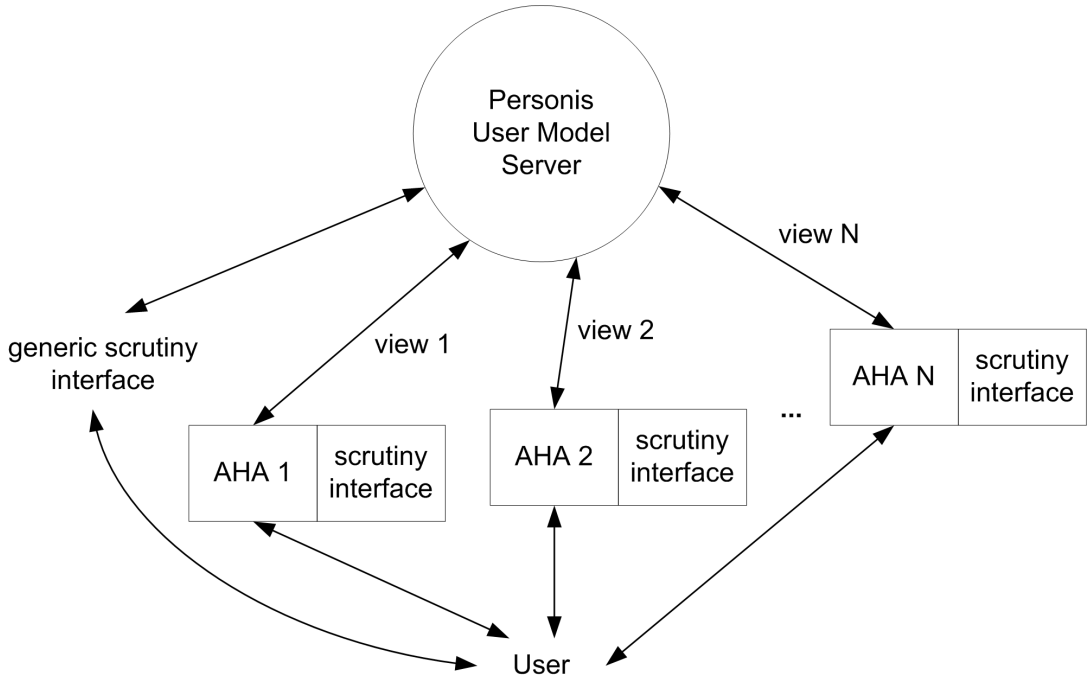

Figure 5.4: Personis Collaboration Architecture [Kay *et al.* 2002]

The privacy issue is addressed by allowing the user to define, which applications have access to particular components of the user model. The user can also control the information sources that should be made available to each application. The access control information is stored with the user model. [Kay *et al.* 2002]

The internal architecture of the Personis server is shown in Figure 5.5. The user model information is stored in an object oriented database (OODB), which makes it possible to distribute the data among several connected servers. This allows to allot required processing power and storage space over several machines. The structure of the user model is represented as in the **um** toolkit (see Sub-section 5.3.3). Components describe parts of the user model and are connected to a list of evidence. Every evidence is stored with the information about its source. Contexts allow to structure the components in a hierarchical structure. Further, views are stored in form of objects in the OODB. The object database also holds the access control information in form of access rights for applications and users. Each aspect can be controlled by assigning read and write rights.

The communication between server and client operates with remote method calls using the XML-RPC[3] protocol over secure socket layer (SSL). Additionally, the server provides a management interface implemented with HTML, HTTP and SSL protocols and accessible from any web browser. The purpose of the *resolvers* is to interpret the data stored in the generic user model for a specific application. The evidence of a component is interpreted and the value of this component is concluded. It is possible to associate a specialized resolver for a particular application. As shown

[3]See `http://www.xmlrpc.org` for details

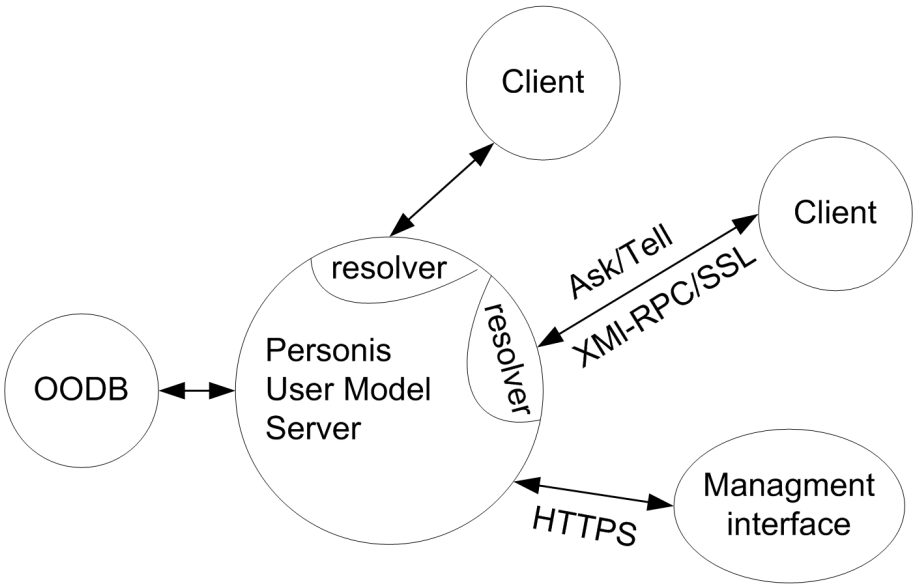

Figure 5.5: Personis Internal Server Architecture [Kay *et al.* 2002]

in Figure 5.5, one client adaptive hypertext system might use one certain resolver. Another client might use another and therefore these two clients interpret the same component differently. [Kay *et al.* 2002]

To summarize, the Personis user modeling server uses the **um** toolkit as the underlying conceptual foundation for the user model. Additional features like scrutiny interfaces, views, OODB and resolvers allow the usage of Personis as a versatile user modeling server. Interesting aspects of Personis are scrutable user models, privacy of user information and the possibility of distributing the user modeling server over several computers.

5.4.3 LDAP-based User Modeling Server

Most of the described user modeling servers so far, do not focus on storage and data handling. An exception is the user modeling server based on the Lightweight Directory Access Protocol (LDAP), which was published by [Fink 2003]. This modeling server uses a directory structure, namely LDAP, to store and manage data about the users. LDAP directories can manage information that is spread across a network of servers by linking this information through referrals.

The user modeling server is based on an LDAP directory server that is complemented by several pluggable user modeling components and can be accessed by external clients, as shown in Figure 5.6.

The core of the architecture is the *Directory Component* with its sub-systems *Communication*, *Representation* and *Scheduler*. The *Communication* sub-system handles the communication with external clients of the user modeling server and with the *User Modeling Components* which are internal clients of the *Directory Component*. Each *User Modeling Component* performs a dedicated user modeling

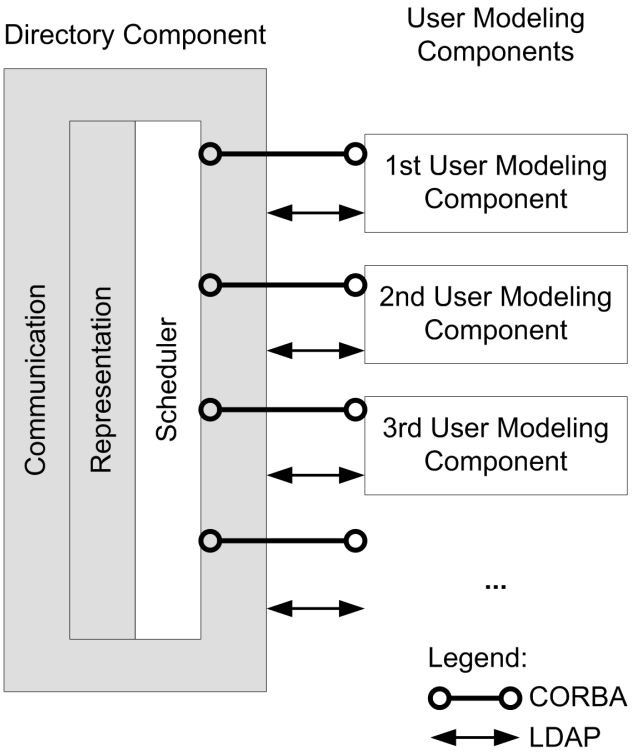

Figure 5.6: Overview of the LDAP-based User Modeling Server Architecture [Fink 2003]

task, such as for example domain-based inferences. The task of the *Representation* sub-system is to manage the directory contents, which is mostly information about the user. Main tasks of the *Scheduler* are to wrap the underlying LDAP server (marked in gray in Figure 5.6) with a component interface and as an interface between the different sub-systems and components of the user modeling server. The *Directory Component* and the *User Modeling Components* communicate via CORBA and LDAP. The *User Modeling Components* perform specific user modeling tasks. The amount of these components is not restricted. A proposed composition of components by [Fink 2003] describes the utilization of a statistics-based *User Learning Component*, a similarity-based *Mentor Learning Component* and a rule-based *Domain Inference Component*. The *User Learning Component* learns user interests and preferences from usage data, and updates individual user models. The *Mentor Learning Component* predicts missing values in individual user models from models of similar users and the *Domain Inference Component* infers interests and preferences in individual user models by applying domain inferences to user information that was explicitly provided by users or implicitly inferred by the other learning components. [Fink 2003]

The possibility to add self-developed *User Modeling Components* enables applications to easily use their own modeling components. The component-based architecture and the defined interfaces (LDAP and CORBA) facilitate the development

and use of such modeling components. Further, the component-based architecture allows to distribute the complete modeling server over several platforms and machines. This is an important aspect concerning data management and performance if this user modeling server is issued for a huge amount of users.

5.4.4 Web Service and Agent-based User Modeling System

At present, a tendency to use Web Services in personalized system can be identified. An example for a service-based personalized system is published in [González *et al.* 2005]. In this approach, Web Services are used to implement agents. The complete system is composed by several agents, where every agent has its own task. Figure 5.7 shows the architecture of this system.

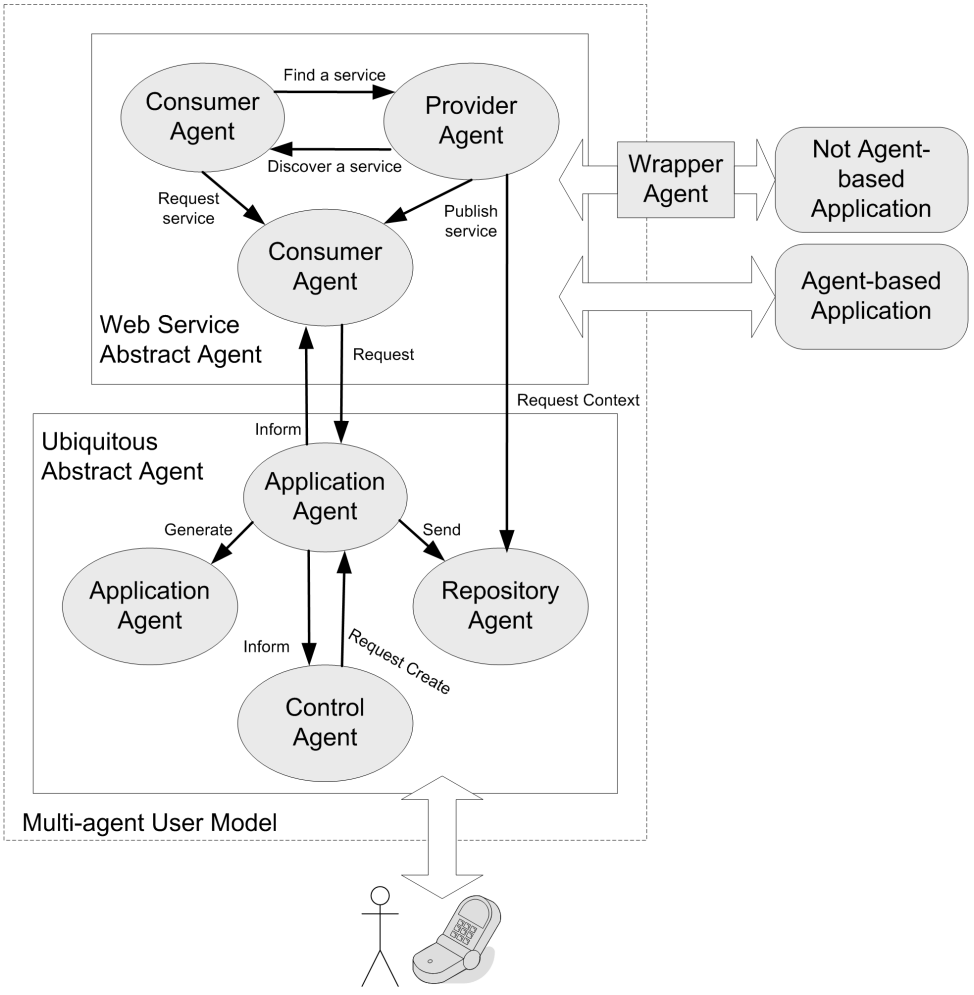

Figure 5.7: Architecture of an Agent-based User Modeling System [González *et al.* 2005]

The user modeling system consists of several agents and can be viewed with different detail levels. In the highest level the system is separated into two agents.

First, the *Web Service Abstract Agent* (WSAA) provides capabilities of automatic discovery of services in the Internet for the user. It can communicate with agent-based and non agent-based applications. When an application is not agent-based, a *wrapper agent* is used for the communication. Second, the *Ubiquitous Abstract Agent* (UAA) provides initialization, identification, interoperability, control, coordination and management of the user preferences allowing a flexible and autonomous human-agent interaction.

There are basically two ways how the WSAA and the UAA can communicate. First, the WSAA requests information from the UAA how to deal with a particular application. Second, the UAA receives information from the WSAA regarding the success or failure of the application interaction. This information is a kind of feedback and is used by the UAA to learn about the user interests. Both abstract agents are designed to be implemented in a distributed environment. Further, the UAA can be used on a mobile device. [González *et al.* 2005]

The WSAA is composed by three types of agents, namely *Account Agent*, *Provider Agent* and *Consumer Agent*. The *Account Agent* maintains a list of applications, which can be connected to the user modeling system. The *Provider Agent* uses information about the context and the *Repository Agent* from the UAA to find new interesting or possible applications. A user requested application is searched and discovered by the *Consumer Agent*. Further, the *Consumer Agent* communicates with the *Provider Agent* and creates appropriate *Application Agents* for each possible application. [González *et al.* 2005]

The architecture of the UAA is divided into four types of agents, namely *Control Agent*, *Creator Agent*, *Application Agent* and *Repository Agent*. The *Control Agent* has three tasks:

- user login,
- dialogue with the user regarding interaction with an application and
- request the *Creator Agent* to generate an *Application Agent* to manage the application.

Tasks of the *Creator Agent* are to acquire the user profile, deliver this information to the *Repository Agent*, generate *Application Agents* and register applications with the usage of the *Control Agent*. An *Application Agent* is dynamically created when an interaction between a user and a particular application takes place. The number of *Application Agents* varies from user to user. The last agent, the *Repository Agent* provides database storage procedures to save the information about the user represented in the user model. [González *et al.* 2005]

A very interesting aspect of this multi-agent user modeling system is its possibility to work in a distributed environment. Further, this environment can also include mobile devices. To provide such an interoperability, the agents are implemented by using Web Services. This technology (i.e. set of standards) allows to spread the system over several different computers and devices.

The described user modeling servers in this section are only some examples among many others. There are further user modeling servers with a commercial utilization

like Group Lens [Konstan *et al.* 1997], Personalization Server [ATG 2005] or Learn Sesame [Caglayan *et al.* 1997]. Commercial modeling server are mainly applied for adapting web sites of e-commerce applications and therefore based on Web Service technologies. The utilized user modeling methods within these solutions are collaborative filtering algorithms, rule-based methods, stereotyping and sometimes overlay methods, which are described in Section 3.5.

User modeling servers make it possible to use a centralized source for information about a user. Such a server can be used by several application systems not necessarily running on the same platform. A distributed application system architecture may use a modeling server within a network. This emphasizes the role of privacy and security. Further, since many different application systems are using the same user modeling server, aspects like performance, scalability and extensibility become important and are addressed by different solutions. For example, Personis (see Sub-section 5.4.2) allows to distribute the data over several platforms and supports privacy by allowing to set privacy tags for each part of the user information.

User modeling standards, which are described in Chapter 4 are rarely supported by user modeling servers. One example for a standard-based user modeling system is OntobUM [Razmerita *et al.* 2003]. OntobUM uses IMS LIP (see Section 4.5) to structure the explicit user model. Additionally, IMS LIP is extended by a behavior concept, which describes characteristics of users interacting with an application system. OntobUM was developed for a knowledge management system to provide personalization, expertise discovery, networking, collaboration and learning.

The following section gives a summary of this chapter and emphasizes interesting facets of user modeling systems.

5.5 Summary and Conclusion

In the beginning of this section a short description about the history of user modeling systems is given. This emphasizes that user modeling is not a "young" topic but has gained popularity in the last years because of the usage in recommender systems.

The user modeling system GUMS (see Sub-section 5.2.1) can be seen as the first application-independent system. It focuses on a versatile usage by applying stereotyping and rule-based methods. Together with the user modeling shell systems described in Section 5.3, they represent the base for the nowadays popular user modeling servers. The modeling techniques applied by the user modeling shell systems are the same as those used in user modeling servers. For example, the **um** modeling shell system (see Sub-section 5.3.3) is used by the Personis server (see Sub-section 5.4.2). Personis embeds the **um** shell with a server structure combined with a database storage.

The present development efforts affect mainly user modeling servers. There are intentions to provide scrutiny for user models (see Sub-section 5.4.2). A scrutable user model allows the user to access and to modify an own user model. Further, intentions can be identified to optimize the data storage, as described in Sub-section 5.4.3 or to provide interfaces to enhance the user modeling system by adding user

modeling components. In this case, a user model is distributed over several sub-models. A sub-model describes a particular "part" of the user (for example demographic data or learning styles). One sub-model is offered by one component. This can be achieved by using a component-based architecture. It is simply possible to add further sub-models by developing and adding components which implement the required sub-models.

Other developments focus on the usage of service-based architecture, which is mainly done by using Web Services (see Sub-section 5.4.4). According to [Tsalgatidou and Pilioura 2002], the advantages of a service-based architecture are:

- easy and fast deployment,
- interoperability,
- just-in-time integration and
- reduced complexity by encapsulation.

New services can be developed by reusing or combining existing services. This allows an easy and fast deployment of new functionality. Since the interfaces of services are well defined and different communication protocols are available by the used framework, any service can interact with other services. By limiting what is absolutely required for interoperability it is possible to develop services which are truly platform- and language-independent. This means that developers do not need to change their development environments in order to produce or consume services. Furthermore, by allowing legacy applications to be exposed as services, the service-based architecture easily enables interoperability between legacy applications or between services and legacy applications. Service-based systems are using discovery mechanisms to find and arrange available functionality. Thus, such systems are self configuring and allow a just-in-time integration of new applications and services. A very important point is the reduced complexity by encapsulation of components, where components are packed into services. This reduces the importance of how a component is implemented and lies the focus onto the behavior of the component. This reduces system complexity, as application designers do not have to worry about implementation details of the services they are invoking.

To be able to combine the advantages of a component-based user modeling system with the advantages of using a service-based architecture an appropriate software development method must be found. Taking user modeling components and packing them into services leads to the usage of a service-oriented architecture. Therefore, an appropriate system architecture should use a service oriented framework. Different service-oriented frameworks are described in Section 6.2.

Concerning user modeling standards, only some user modeling systems (for example OntobUM [Razmerita *et al.* 2003]) work with the standards described in Chapter 4. In such cases, the stored data is arranged according to the standard specification. In many user modeling systems, the user model data is stored in a self-developed schema. This is a problem if two different user modeling systems are

compared. For every system, the gathered data about a user has to be extracted from the user model.

The following chapter describes basic thoughts related to the solution approach proposed by the author of this thesis. The background knowledge about different topics is gained from the topics examined so far in this document, like user modeling techniques (see Chapter 3), available standards (see Chapter 4) and the described systems in this section.

6. Basic Reflections for the Solution Approach

6.1 Introduction

This chapter depicts different points of interest which must be clarified before starting with the proposed solution for this thesis. The first question to be answered is the optimal technology to develop the user modeling system (see Section 6.2). Secondly, the topic privacy and security for user modeling systems is examined concerning available standards, techniques and possible solutions (see Section 6.3). Finally, the necessary levels and the organization of learner profiles and learner models are described (see Section 6.4). In the last section, the conclusion states the outcome in form of a recommendation for the solution approach.

6.2 Service-oriented Architecture

The examination of different user modeling systems (see Chapter 5) recommends the utilization of a service-oriented architecture (SOA) for the implementation of the proposed solution, which is described in Chapter 8.

This section starts with a description of SOA. The terms, concepts and components, which are related to SOA are depicted. After this introduction into SOA, Web Services, services-oriented framework and the principles of service-oriented software design are described (see Section 6.2.1 - 6.2.3).

SOA has become popular in the last few years due to Web Services. The motivation to develop another programming paradigm was the insufficiency of object-oriented architecture to be able to reuse functionality and not just code. Further motivations were the needs to deal with distributed software, application integration, varying platforms, varying protocols, various devices, the Internet, etc. Additionally to object-oriented software design, component-based architecture is used by SOA to provide reusable functionality. Instead of replacing these two well known and proofed software development techniques, SOA merges them and combines their advantages. This is done by following the application architecture schema shown in Figure 6.1. There are less details of the implementation visible as one gets closer to the user of the application who interacts with the services and therefore does not need to know anything about the used objects or components. [Hashimi 2003]

Along with SOA comes a new vocabulary describing the key concepts. According to [Hashimi 2003], this new terminology embraces concepts, such as the following:

- service,
- message,
- dynamic discovery,
- and Web Services.

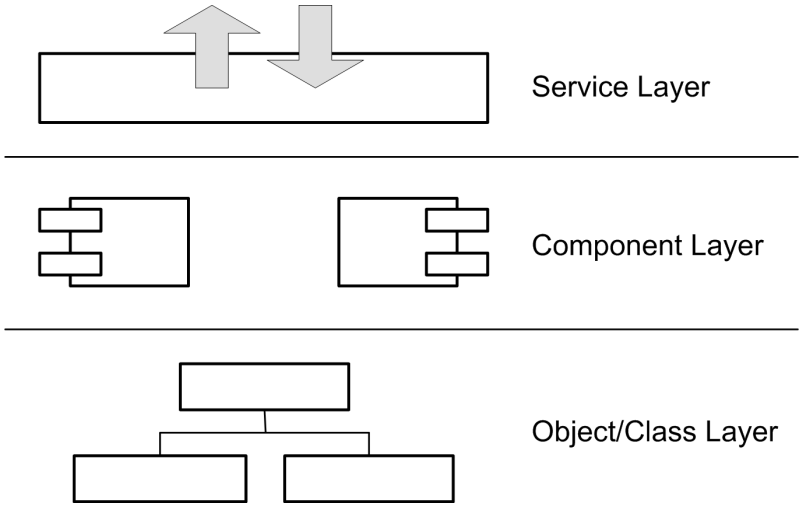

Figure 6.1: Application implementation layers: services, components, objects [Endrei *et al.* 2004]

Service: A *service* in SOA represents a unit of functionality. Further these *services* fulfill three conditions. First, the interface for the service is platform-independent. Second, the *service* can be located and invoked dynamically and third, the service has and maintains its own state and is self-contained. [Hashimi 2003]

Due to the platform-independence, the service consumer is able to connect and use the *service* independent of the location, the operating system and the programming language. The concept of *dynamic discovery* states that a discovery service (for example, a directory service) is available (see Figure 6.2). Such a directory service provides a look up mechanism to service consumers where they can find a service-based on given criteria. For example, consumer A is looking for a database access service. In this case consumer A might connect to the directory service and might get a list of available services which offer connections to a specific database.

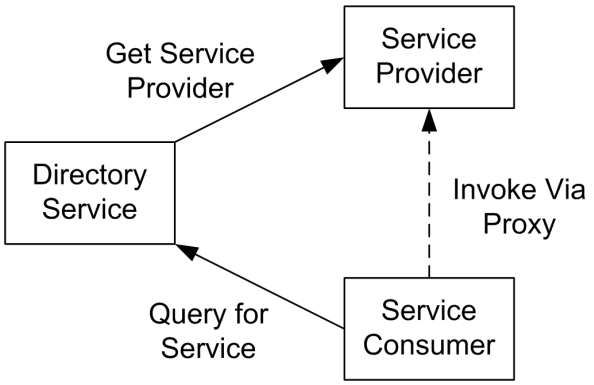

Figure 6.2: Directory service [Hashimi 2003]

Message: The communication between service providers and consumers is established by using *messages*. Every service publishes an interface contract. This contract defines the behavior of the particular service, its acceptable messages and the possible return messages. Since the interface contract must be platform- and language-independent it is necessary to apply a technology which is not restricted to a specific language or platform. In most cases XML is used to construct these *messages*. XML offers the required versatility to clearly define messages. [Hashimi 2003]

Dynamic Discovery: At a high level of abstraction, SOA consists of three core parts, namely service providers, service consumers and the directory service. The roles of service providers and consumers are obvious but the task of the directory service needs some further explanation. The directory service is used to mediate between service providers and consumers. Service providers are registered in the directory service and the service consumers are now able to query the directory service to find the appropriate service provider. Utilizing a directory service provides a *dynamic discovery* of available services. This allows to decouple consumers from service providers and therefore the service consumer is able to choose the service provider at runtime. [Hashimi 2003]

Web Services: *Web Services* are not an underlying concept of SOA. They just represent one kind of realization of the SOA concept. Nevertheless, SOA has become very popular because of *Web Services*. *Web Services* are described in more detail in Section 6.2.1.

After explaining the terms which are related to SOA , two different types of service-oriented design approaches are identified. First, the Web Service concept (see Section 6.2.1), and second, service-oriented frameworks (see Section 6.2.2). The following two sections focus on these two approaches.

6.2.1 Web Services

As already stated, Web Services are a relatively new technology and are accepted as an implementation of service-oriented architecture. The reason for this fast gain in publicity is the ability to provide a distributed computing approach by integrating extremely heterogeneous applications over the Internet [Endrei *et al.* 2004]. This integration relies on the complete independency of programming language, operating system and hardware of Web Services. The World Wide Web Consortium (W3C) works with the following definition of a Web Service:

> *"A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with*

an XML serialization in conjunction with other Web-related standards." [Booth *et al.* 2004]

As this definition implies, a Web Service is mainly based on open technologies such as:

- eXtensible Markup Language (XML)
- Simple Object Access Protocol (SOAP)
- Universal Description, Discovery and Integration (UDDI)
- Web Services Description Language (WSDL)

Considering the W3C definition of Web Services and the above mentioned open standards, a Web Service is basically a SOA service with at least two additional constraints. First, the interfaces must be based on Internet protocols such as HTTP, FTP and SMTP. Second, the messages must be in XML format, except for binary data attachment. However, using this open standards and fulfilling these restrictions allow also interoperability between different company solutions, which represents one of the reasons why Web Services have become so popular.

An illustration of operations, the components providing them and interactions used by the concept of Web Services are shown in Figure 6.3. Basically, the figure represents the same concept as shown in Figure 6.2 except that interactions and components are restricted to the definition of Web Services.

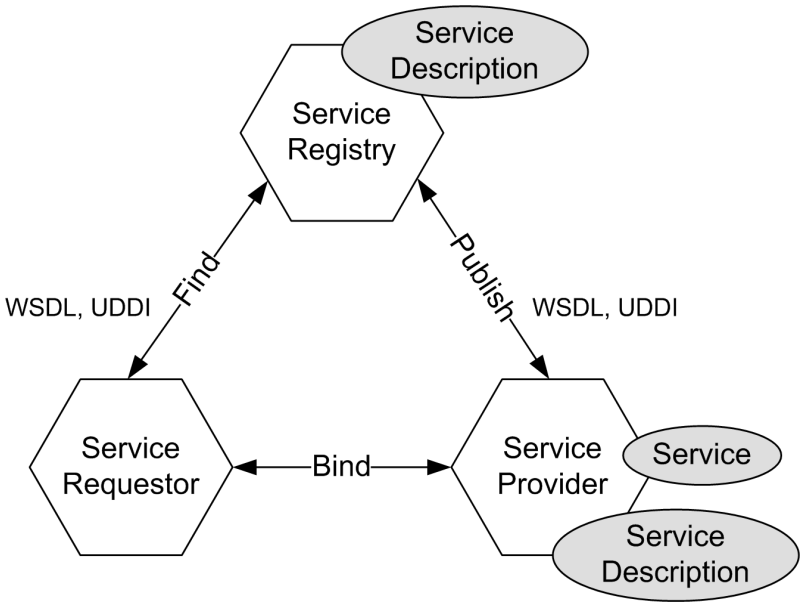

Figure 6.3: Web Services roles, operations and components [Kreger 2001]

The *service provider* is the platform that hosts access to the service while the *service requestor* or *consumer* is looking for and invoking or initiating an interaction

with a service. The *service requestor* role can be played by a browser driven by a person or a program without a user interface, for example another web-service. The *service registry* is a searchable registry of service descriptions where *service providers* publish their service descriptions. *Service requestors* are able to find services and obtain binding information. *Publish*, *find* and *bind* are the operations, which are performed by the different roles. The service provider must *publish* the particular service so that the *service requestor* is able to *find* this service. In the *bind* operation the *service requestor* invokes or initiates an interaction with the service at runtime using the binding details in the *service description* to locate, contact and invoke the service. The standards WSDL and the UDDI are used to *find* and *publish* services. [Kreger 2001]

The Web Service technology is the logical progression of object-oriented and component-based design patterns. With their loose coupling between functional components and usage of open standards, it is possible to develop software which uses today's ubiquitous WWW. But applications are often used within closed networks like the Intranet. Therefore it is necessary to distinguish between SOA and Web Services. Web Services implement the concept of SOA within the environment of the WWW.

Before starting with the development process of a service-oriented application it is useful to look for already available frameworks. By applying a service-oriented framework the development process can be reduced by the already available services for communication, security, lookup, and so forth. The following section describes available frameworks which allow to develop service-oriented applications.

6.2.2 Service-oriented Frameworks

Service-oriented frameworks offer an environment to deploy and run self developed software in a service-oriented manner. There exist numerous frameworks for many different purposes, platforms and operating systems. Basically, every framework works with components which represent a part of the functionality and pack this component in a service. Besides several commercial initiatives and specifications, the following section focuses on OSGi and Openwings. For both of these framework specifications there exist free available implementations.

OSGi

OSGi stands for *Open Service Gateway initiative* and is a coalition of more than 70 companies which developed a common open standard for service gateways. The OSGi specifications define an environment, which allows to run services. These specifications are currently available in version 3, which was published in March 2003 [OSG 2003].

The core component of the OSGi specifications is the OSGi framework. The framework provides a standardized environment for applications (called bundles). The framework is divided in a number of layers (see Figure 6.4):

- Execution Environment
- Modules
- Life Cycle Management
- Service Registry

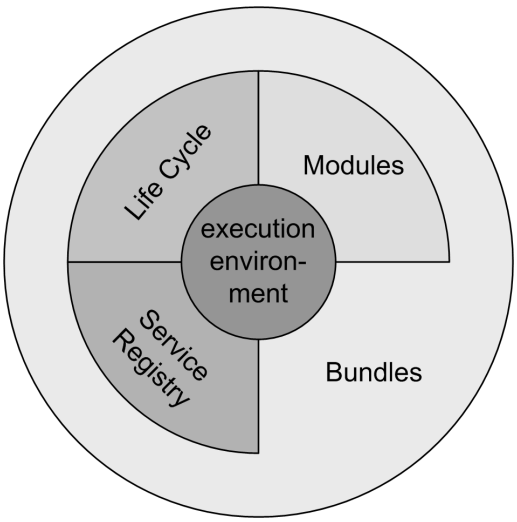

Figure 6.4: Architecture of the OSGi framework [OSG 2003]

Additionally, a security system is contained, which is available for all these layers. This security system is based on the Java 2 security model.

The *execution environment* is the specification of the Java environment. For example, all Java 2 configurations and profiles (like J2SE) are allowed *execution environments*. The *Modules* layer defines the class loading model. OSGi is based on Java but is extended by modularization. Therefore, compared to Java, where normally a single classpath contains all the classes and resources, by means of the OSGi *Modules* layer it is possible to add private classes for a module as well as controlled linking between modules. The *Life Cycle* layer adds bundles that can be dynamically installed, started, stopped, updated and uninstalled. Bundles represent a software component and rely on the module layer for class loading. The *Service Registry* provides cooperation for bundles. Bundles can cooperate by using class sharing techniques or by using the *Service Registry*, which allows to share objects between bundles. There are several events defined to handle the life cycle of services. For example, if a service is started the `started` event is thrown. [OSG 2003]

Since OSGi represents the specification of a service-oriented framework, there exist several implementations of this specification. Beside many commercial implementations there are some free and open source implementations available, namely JEFFREE[1], Oscar[2] and Knopflerfish[3].

Openwings

Openwings[4] was founded by General Dynamics Decision Systems[5] and Sun Microsystems[6] in the year 1999. The current section gives an overview over the Openwings architecture and its basic concepts.

Openwings defines its framework as follows:

> *"Openwings is a set of open systems specifications for a framework that enables the development of highly available, secure, distributed systems for mission critical applications. The initial implementation of this framework utilizes Sun's Jini technology to provide ad-hoc integration of system components as well as increasing the interoperability in a 'systems of systems' environment."* [Openwings 2003]

In other words, Openwings is a service-oriented framework which enables the development of components. Components are independent from each other and are able to consume or provide a service. The communication between components follows the client-server concept where common contracts specify the form of communication.

The Openwings architecture (see Figure 6.5) was designed to enable the development of a service-oriented network-based system. Such systems are independent from middleware, databases and platforms. The base is represented by the *Component* service. A *Component* service offers all needed commands to handle a service. *Connector* services are used to establish connections between components (between a service provider and a service consumer). *Install* services handle the installation of components. To be able to unite several services it is possible to use the *Context* service. The same properties and policies are assigned to every service within a context. A policy describes a component. There can exist several policies for one component, e.g. a security policy or an installation policy. The security in Openwings is covered by offering three different types of *Security* services, namely *Code*, *Transport* and *Service security*. [Bieber and Carpenter 2001]

A reference implementation of the Openwings specifications is currently available in version 1.1 but not as open source. Openwings is described in more detail in Chapter 7.

As described in this section it is likely to use a service-oriented framework for the development of service-oriented software. This saves a lot of development time

[1] http://jeffree.objectweb.org/
[2] http://oscar.objectweb.org/
[3] http://www.knopflerfish.org/
[4] http://www.openwings.org/
[5] http://www.generaldynamics.com/
[6] http://www.sun.com

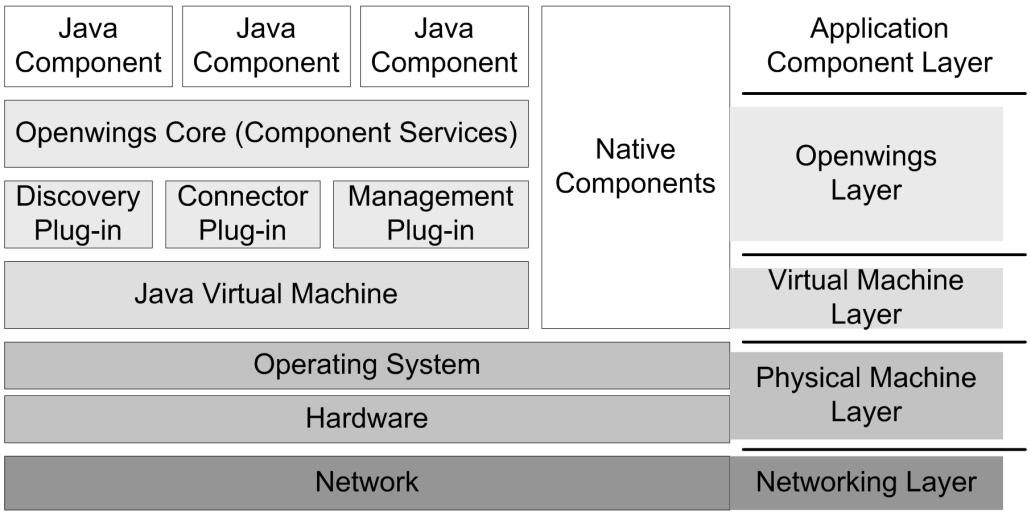

Figure 6.5: Openwings Architecture [Bieber and Carpenter 2001]

and resources even though the framework may not realize exactly all the features which would be ideal for a particular software project. Therefore, an appropriate framework has to be chosen. After a framework has been chosen, the process of software design can be started. Since SOA does not come along with well-defined software development patterns, the next section describes rules and ideas on how the software design process may be carried out.

6.2.3 Service-oriented Software Design

As already stated in Section 6.2, SOA is based on software components. This means that at least one software component is packed into a service. A service can simply be defined as

> *"... a group of related components that carry out a given business process function."* [Romand *et al.* 2005]

Thus, SOA focuses on the development of services rather than small components. These services provide a higher level of abstraction from a functional point of view. Further, the motivation to develop service-oriented applications is that the parts of these applications (services) can be reused by other application. The smaller a service is and the less functionality is packed into a service the more a service is reusable. On the other hand, dealing with a whole bunch of small services increases the complexity and the maintenance affords to keep the application running. Therefore, it is necessary to consider several parameters within the design process when the functionality of services and the components are defined. This point is taken for further investigations and two different, contradictory solution approaches are defined in Chapter 8. The first approach is following the concept of packing one component into one service which leads into many services. The second approach defines logical and functional units, and puts them together into one service.

As already stated at the beginning of this section, SOA has become popular by the introduction and usage of Web Services. But often there is a misinterpretation if these Web Services are considered as the same as SOA. Web Services are only one of several possible realizations of SOA concepts based on the WWW infrastructure. Other realizations of SOA are implementations based on service-oriented frameworks. These frameworks are valuable since they offer an environment where developers can focus on the functionality and do not have to care about realization of service-communication, -lifecycle, -discovery, etc.. But the concern about good service-oriented software design remains. Beside the guideline that a service should be able to represent one business process and should consist of components, the software designers can choose the granularity in which components and therefore services are realized.

In the next section further important points concerning user modeling systems, namely privacy and security, are depicted.

6.3 Privacy and Security in User Modeling

Privacy and security are two important topics in the field of user modeling. Since user modeling is based on the collection of very intimate and personal data about users it is necessary to protect this information from unauthorized access of systems and persons.

Security in user modeling is often seen as not a goal itself but as a precondition for realizing privacy [Schreck 2001]. Nevertheless, the following section describes privacy and security as separated topics because there are different problems and solutions to ensure privacy and security.

6.3.1 Privacy

The term privacy was initially defined in 1890 by a judge called Louis Brandeis. He specified privacy as an individual's right to be left alone. This definition is to weak for nowadays application, therefore a better definition of privacy is stated by [Westin 1970] as follows:

> *"Privacy is the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others."*

Therefore, concerning a user modeling system, it is necessary that a particular user is able to adjust his level of privacy by selecting the information which should be shared and to whom. To realize this requirement a well thought system is needed to offer privacy to the user.

According to [Schreck 2001], the confidentiality of user model information can be achieved by processing user model information anonymously or pseudonymously. The user model information is thus no longer administered as data, which is connected to a user but it remains interpretable and usable for connected application systems.

There are several levels of anonymity. Depending on the application of the user model an appropriate level of anonymity must be selected. [Schreck 2001] lists six different levels of anonymity ranging from the unambiguous assignment of data to a person to the complete disengagement of data from the person:

- super-identification,
- identification,
- latent identification (controlled pseudonyms),
- pseudonymous identification (uncontrolled pseudonyms),
- anonymous identification, and
- anonymity.

With *super-identification* the user is identified using a third-party, which resides outside the system. This guarantees that no system component is able to counterfeit the identity. In the level of *identification* the user identifies himself by demonstrating knowledge of secret (for example a password) which is compared to the stored value. By the procedure of *latent identification* or *controlled pseudonyms* the user identifies himself to the system and obtains one of the predefined pseudonyms. This procedure is often used in box number advertisements. Using *pseudonymous identification* or *uncontrolled pseudonyms* methods let the user enter a unique pseudonym and a secret when using the system for the first time. Afterwards, the pseudonym and the secret are used for all subsequent sessions. The system is unable to clarify the identity of a particular user and thus it is also unable to link the pseudonym to the user's identity. This method is used in most web-based services. Through *anonymous identification* the user gains access to the system by providing a secret without revealing his identity. Thus, the system is unable to distinguish between users which share the same secret. At the highest level of anonymity, the user does not identify nor authenticate himself to the system. The system is not able to differentiate between the users. [Schreck 2001]

Comparing the requirements of user modeling with the levels of anonymity, the *pseudonymous identification* is the best compromise between privacy demands and the requirements of user modeling. Pseudonyms also make it possible to link a user model and the user being modeled without revealing the user's identity to components of the user adaptive system or to the user modeling system.

Anonymity and pseudonymity offer advantages for user modeling systems by limiting the relationship between persons and their personal data. Further, by offering the user the demanded level of privacy the acceptance of personalized systems increases. [Schreck 2001]

Beside the privacy which is needed to prevent unauthorized persons or systems of getting information about a specific person, the second issue in this section is the security of user information.

6.3.2 Security

Security deals with the protection of the information itself. Compared to privacy where access to the information in a user model is limited by the privacy settings, security concepts are needed to prevent unauthorized access of systems to user modeling systems and data.

The security can be sustained by applying different concepts, namely *Denial of Access* or *Selective Access* [Schreck 2001]. There are two ways how *Denial of Access* can be interpreted. First, as a *denial of access* to the connection between the user and the stored data, and second, as *denial of access* to the information of a particular user. Techniques to provide denial of access to data are anonymity (see Section 6.3.1) and encryption. Anonymity cuts the relation between the particular user and the information about him. To protect personal data from inspection when it is exchanged between the user model and its clients, the information must be encrypted.

Technologies for data encryption are widely known and applied. The discussion of such technologies would exceed this work. Thus, in the following section only the technologies and guidelines which are available to ensure privacy in e-learning systems are described.

6.3.3 Privacy Technologies

Beside the importance of applying privacy techniques for e-learning systems it is also necessary to describe this level of privacy. According to [El-Khatib *et al.* 2003] a user of an online-learning environment has *concerns* about privacy, like what information is gathered and for which purpose. Further *interests* are how long this information is kept and if this information is revealed to other companies.

To address these questions the W3C developed the Platform for Privacy Preferences Project (P3P) [Cranor *et al.* 2002]. P3P allows web sites to described their privacy policies in a standardized form, which can be automatically retrieved and interpreted by the user client. These privacy policies are arranged in a machine-readable XML format. The user can specify privacy requirements and the web site can publish its privacy policies, both according to the P3P form. Taking the requested privacy requirements of the user, the user client can automatically make decisions regarding the acceptability of the provided privacy level of the service. If the user's privacy requirements are not fulfilled, the user is warned in form of a notification. [El-Khatib *et al.* 2003]

According to [Kobsa 2001b], personalized systems (e.g. adaptive e-learning systems) should support P3P to allow users to express their need of privacy. But P3P is far away of being able to described all privacy preferences regarding personalized systems.

Concerning standards like PAPI and IMS LIP (see Chapter 4), they also address concerns about privacy. PAPI does not specify a detailed model or technology and no particular privacy policy is specified, but a valuable feature facilitating privacy protection is defined. This feature is the logical division of learner information, which allows to partition learner information and assign different privacy policies to

each of these parts of information. Similar to PAPI, IMS LIP treats data privacy as an essential requirement but does not define any details on the implementation mechanism or the architecture.

The purpose of user modeling systems is to gather and process as much information about a user as required and possible. This is necessary to provide personalized systems the needed user information. Since such, often very personal, data is stored centralized in a user model, concerns about privacy and security have to be taken seriously and the applied solution must be well thought and enable the privacy needs of the users. Privacy and security mechanisms must protect the user modeling system from unauthorized access of systems or persons and from intrusions into the modeling system or the communication.

Anonymity and pseudonymity provides in most cases enough privacy since a user modeling client is not able to query information about a user from whom it does not know his pseudonym and his secret. The user is in this case responsible for the decision to whom she/he can safely give away the user data. In order to offer the user more transparency and information about applied privacy methods, the P3P is in this case a good initiative.

In the next section the modules and the relevant information is described, which are necessary for an exploitable user profile.

6.4 Levels of Profiling the Learner

As described in Section 2.2, profiling is used to gather and organize information about a user in form of raw data. At this state, there is no interpretation performed on the received data. The idea of profiling is based on a useful logging of user interaction with the system. To be able to extract needed information from the data stored in the user profile it is necessary to apply a particular user model. Such a user model takes the data from the user profile, interprets it and models a particular aspect of the user. To enable this information enrichment it is necessary to provide the user model with the needed data. Therefore, the user profile must gather and store relevant data about the user. The current section describes and examines an idea of how the data in the user profile can be structured and organized.

Initially, it is necessary to identify the information sources from where data for the user profile is received. Having knowledge about these sources and the kind of information which is contained within these data, allows to organize the content of the user profile. For example, if the information source is an eye-tracking device, the received data contains information about the gaze movements on-screen.

The following sub-sections describe ideas about how the data of the user profile and the user model are organized.

6.4.1 Organizing the Learner Profile

In the field of adaptive e-learning it is necessary to gather information which allows the personalization of (in most cases) the learning content. Regarding the learner

profile, data about shown instructions, interactions with the system and personal details are important. Based on these types of information it is possible to split the learner profile into a *history component* and a *learner information* component, where the *learner information* covers all personal details about the learner and the *history data* records instructions shown to the learner and interactions of the learner with the system.

The *learner information* unit does not need much logical sophistication since all related values are in the form of key-value pairs, for example, first name, age, etc. The *history component* is more complex and must organize interactions and instructions in a way so that the learner model is able to process and interpret this data. The learner profile shown in Figure 6.6 depicts the structure of the proposed solution for a modularized learner profile.

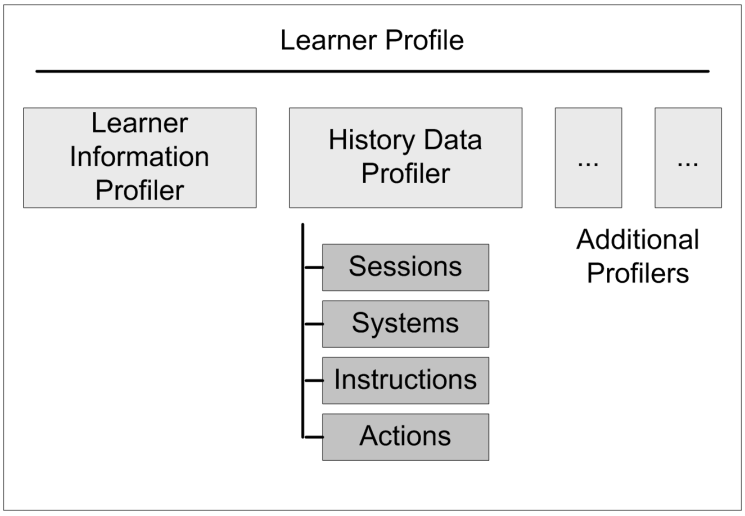

Figure 6.6: Structure of the Learner Profile

The learner profile which represents the highest level is separated into a *Learner Information Profiler* and into a *History Data Profiler*. This separation into components allows to add *Additional Profilers* according to the application requirements. The *History Data Profiler* is sub-divided into a *Session*, *Instruction*, *Action* and *System handler*. The *Session handler* assigns every session of a particular learner a unique id. All occurring actions and instructions are assigned to one particular session. Used systems are handled by the *System handler*. Systems are for example, a course, a forum or a chat environment. Within a session, there might be several systems but an action or an instruction is connected to a particular system. At last, *Instruction* and *Action handler* record the received data about instructions and interactions.

6.4.2 Partitioning the Learner Model

Concerning the learner model, two different types of information are relevant, namely domain specific and domain independent information (see Section 3.4). Domain spe-

cific information is connected to the knowledge domain, which in turn represents a context-dependent conceptual space. At the level of the underlying learner profile, domain specific information is received when the application sends information about interactions with the system related to the learning content. For example, information about the shown instructions or results of an exam. The recording of these interactions is necessary to build a learner model which contains domain specific information. The learner model is needed to enable the adaptation of the learning content according to the progress and the current domain knowledge of the learner. Thus, recording the progress of a learner through the learning material is an essential point. [Conlan *et al.* 2002b]

Domain independent information is the second type of information stored in a learner model. As stated in Section 3.4, domain independent information is necessary to allow adaptation according to the learner's preferences, learning styles, cognitive aptitudes, etc..

Splitting the learner model into several components, where each component represents one particular part of the learner model leads to a learner model consisting of several sub-models. This idea is already depicted and used by the LDAP system described in Section 5.4.3. Every learner modeling component performs specific modeling tasks. In this case, additional learner modeling components can be added or removed according to the requirements of the application.

This section has listed reasons why it is important to consider the organization of learner data. A well structured learner profile allows the learner model to exploit the profile data. Basically, a component-based structure of the profile and of the model is useful and has two main advantages. First, the user data and information can be structured into logical parts, and second, it is easier to add further sub-profiles and sub-models.

6.5 Summary and Conclusion

As may be concluded from an examination of user modeling systems described in Chapter 5, it is advisable to utilize SOA as the underlying solution concept. The first section of the current chapter describes the basic principles of SOA and their implementation in form of Web Services or service-oriented frameworks (see Sections 6.2.1 - 6.2.2). Because of the protocol limitations of Web Services two common service-oriented frameworks (OSGi and Openwings) are described. Openwings has its advantages compared to OSGi, because along with the specification also a free reference implementation is available. For OSGi there are several implementations of its specifications, but only some are freely available. Finally, service-oriented software design is described. Although some rules and patterns for the software design are available, they lack in the recommendation of how big a service should be. This fact motivates to design two different solutions approaches, namely a micro approach (see Section 8.5) where one component is packed into one service, and a macro approach (see Section 8.4), where only a few services embrace the whole functionality.

The Section 6.3 deals with privacy and security for user modeling systems. It emphasizes the importance of privacy and introduces standards, techniques and solutions for satisfying the required level of privacy and security in order to keep the data about the user safe and secure. The P3P is a new initiative of the W3C to standardize some privacy principles and allow the user to define the own privacy requirements. Further, it informs the user about the offered privacy level of an application. Again, the privacy precautions by the user modeling standards, particularly PAPI and IMS LIP are described (for more information about these standards see Sections 4.4 - 4.5).

In the last section of this chapter (see Section 6.4), the levels of learner profiles and learner models are depicted. Concerning the received data by a learner profile it is possible to split the profile into several components. For example, data about learner interactions with the system is recorded in one component while data about shown instructions is stored in another component. Similar to such a learner profile, a learner model can be structured in the same manner. This partitioning allows to build a flexible and extensible learner modeling system.

As the examination of service-oriented techniques within this section results in a recommendation to utilize the Openwings framework for the solution approach, the next chapter describes the Openwings framework in more detail.

7. The Openwings Framework

7.1 Introduction

Motorola (General Dynamics Decision Systems) and Sun Microsystems founded the Openwings consortium[1] in June 1999 as an open community. The goal of the community is to specify a framework which is independent from the used middleware, databases, computer organizations and operating systems. Up to now more than 100 companies have joined the Openwings community [Carpenter and Bieber 2003]. In addition to this specification, also a reference implementation was developed. The Openwings white paper [Bieber and Carpenter 2001] and its specifications are the main source for this chapter. Openwings is a framework for service oriented software development (see Section 6.2 for more information about service oriented software development). It provides abilities to develop software components and deploy them in form of services. The reference implementation represents the topic of this chapter and is used by the solution approaches which are introduced in Chapter 8.

Starting with an overview of the Openwings framework, the component and developing model defined by Openwings is described (see Section 7.3 and Section 7.4) followed by security issues covered by Openwings. At the end of this chapter a summarization is given.

7.2 Overview of Openwings

The architecture of the Openwings framework is layer-based as shown in Figure 7.1 . The Network layer describes the lowest level. It stands for a pile of computers and other devices which communicate with the Component Services (see Section 7.3 for details). Component Services themselves, are able to communicate with each other, even if they are running on different platforms, by using connectors. For the framework it makes no difference on which computer or platform a Component Service is installed and running. The different components are assigned to Container Services, which observe the running component services (see Sub-section 7.2.1).

The next higher layer represents the Java Virtual Machines (JVMs) which are running on the operating system from different Java editions. The restriction which JVM and hence which computer organization and operating system is supported depends on the used discovery plugins and Connector Services. The reference implemenation needs an adequate Java 2 Standard Edition JVM because of the Jini discovery plugin [Jini 1999], RMI [RMI 1997] for synchronous connectors and JMS [Hapner *et al.* 2002] for the asynchronous connectors. It is not necessary that the self-developed services are implemented in Java but they must define Java interfaces. Within the reference implementation the services are also realized in Java.

[1] `http://www.openwings.org`

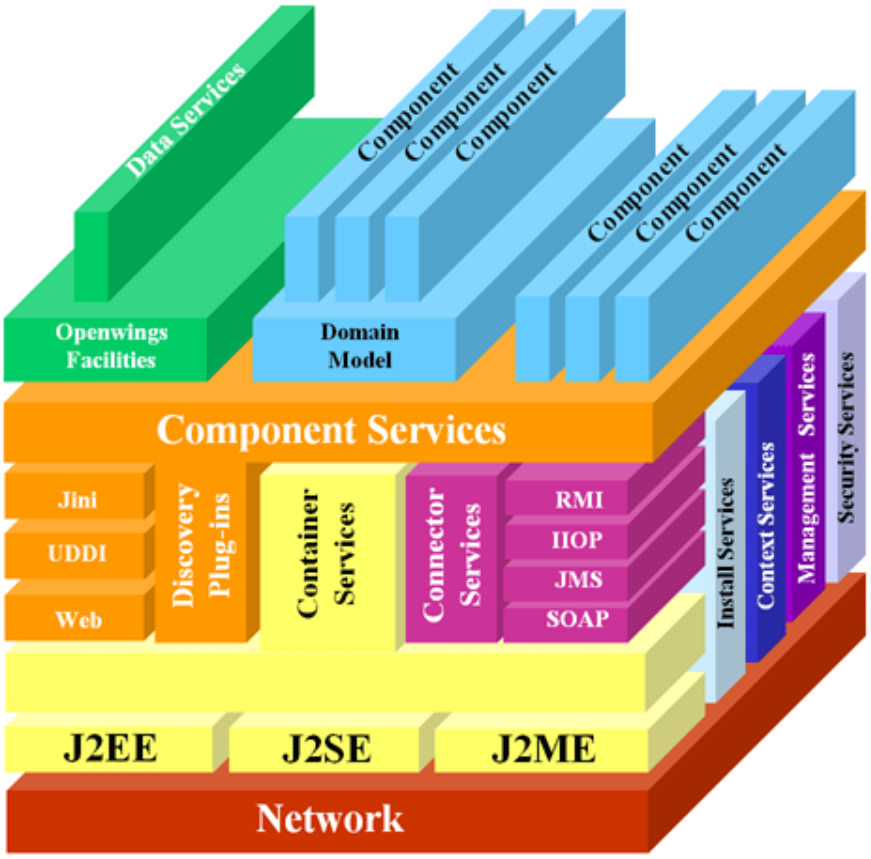

Figure 7.1: Architecture of the Openwings Framework [Bieber and Carpenter 2001]

7.2.1 Openwings Core Services

On the layer above the network layer, the JVM services that enable the usage of the framework are located. Compared to Component Services, these services are provided by Openwings and called Core Services. Core Services are the Container Service, the Connector Service, the Component Service, the Install Service, the Context Service, the Management Service and the Security Service. These Core services are described in the Sections 7.2.1 - 7.2.1.

Every Core Service abstracts from certain used technologies. Services offered by the framework are accessible and can be called from software outside the framework. This external software as well as processes within the Container Service (see Subsection 7.2.1) are able to use Component Services to produce new service objects. Furthermore, they are able to discover and use all services of the framework with the help of Component Services. Therefore the Component Service lies on top of all other services.

In the following the Core Services provided by the framework are introduced. These seven services are the Container Service, the Connector Service, the Component Service, the Install Service, the Context Service the Management Service and the Security Service (see Figure 7.1).

Container Service

By using Container Services it is possible to use different discovery plugins and Connector Services. A Container Service provides a runtime environment for executable components and can be used to discover service objects and to communicate with them. For the service object discovery the discovery plugin is used. Connections for the communication between service object are established by using Connector Services. The specification focuses on the usage of Jini[2], UDDI[3] and Web Services as discovery services. For the implementation of the Connector Services the technologies RMI[4], IIOP[5], JMS[6] and SOAP[7] are taken into consideration, but an extension to use other technologies for the discovery of service object is possible.

Connector Service

Connector Services provide transport protocol independency by abstracting synchronous and asynchronous communications in terms of Java interfaces. Interchangeable connectors handle the details of specific protocols. The connector framework is designed for independent usage from the rest of the architecture in other frameworks, such as Enterprise Java Beans (EJB).

Connector Services specify an API for connectors and provide tools for the generation and location of protocol specific connectors. According to [Bieber and Carpenter 2003], they have the following assignments :

- Automatic generation of connectors for different service interfaces and connection technologies.
- Generation of objects for the communication between service objects (so-called proxy objects).

Service objects take these generated objects to communicate with other service objects. Figure 7.2 shows the anatomy of a connector. A connector is composed of a user proxy and a provider proxy. A connector is installed when each proxy is inserted in the appropriate component.

[2]More information about Jini can be found at `http://www.jini.org/`
[3]Universal Description Discovery and Integration UDDI `http://www.uddi.org/`
[4]Java Remote Method Invocation RMI `http://java.sun.com/docs/books/tutorial/rmi/`
[5]Internet Inter-ORB Protocol IIOP
[6]Java Message Service JMS `http://java.sun.com/products/jms/`
[7]Simple Object Access Protocol SOAP `http://www.w3.org/TR/soap12-part1/`

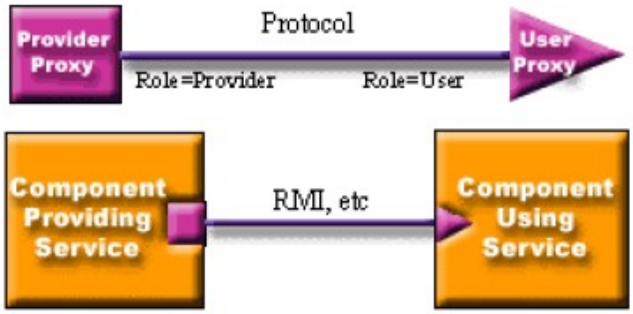

Figure 7.2: Connector Architecture [Bieber and Carpenter 2001]

Component Service

With Component Services it is possible to provide and use basically every software service object in Openwings. Every Component Service is assigned to a Container Service. A Component Service provides the following abilities:

- Creating service objects and making them available within the network.
- Signing in and signing out service interfaces for the communication.
- Creating Connectors which enable the communication with other service objects.

Component developers are only concerned with a small set of operations. For synchronous services this means providing and using services, removing a provided service, and discarding a used service. For asynchronous services this means publishing and subscribing to services, unsubscribing a service, and unpublishing a service (see Figure 7.3).

For complete details on how components are developed, created and handled within Openwings see Sub-section 7.2.2.

Install Service

The Install Service is responsible for the installation of components. A component must own an Installable Component Descriptor to be installable. For details on the Installable Component Descriptor see Sub-section 7.4.5.

The Install Service can act as an application server to make components available to other platforms. With the Install Service it is possible to register a listener for events generated by the service. Such installation events allow for example the Container Services to instantly start components after their installation.

To install a new component, the Install Service first authenticates the component. Then the files of the component are extracted and placed in the appropriate locations on the platform. Next, the Install Service tries to resolve dependencies and policies,

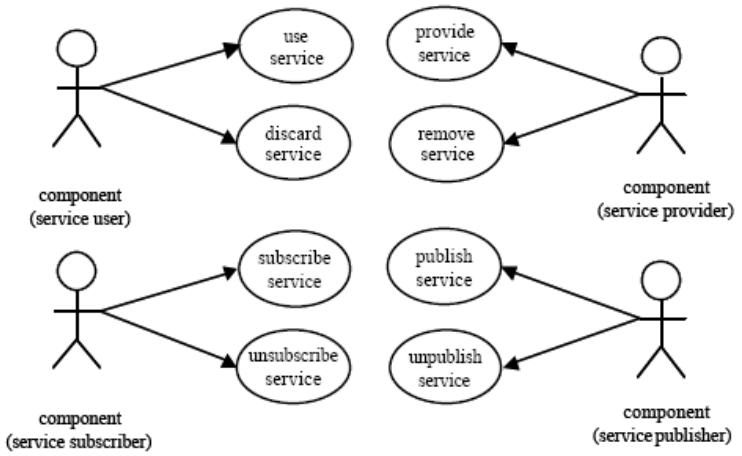

Figure 7.3: Component Services Use Cases [Bieber and Carpenter 2001]

described in Sub-section 7.4.3, for the component. Once the component is in the resolved state it can be executed. If the user chooses to uninstall a component, the component moves into the uninstalled state and its files are removed from the system. Further, the Install Service provides a voting mechanism which allows other components to delay the uninstallation of needed components.

Context Service

Openwings allows to combine several platforms on which Openwings is running into one context. Every context is administrated by its own Context Service. Context Services are discussed in more detail in Sub-section 7.2.2.

Management Service

Management Services offer a connection between Openwings and different software tools and concepts which support management tasks [Smith *et al.* 2003].

The Management Service allows to simply create additional service objects. Each Openwings component contains an extensible management framework that allows management plugins to be added and remotely controlled.

Security Service

Security Services offer code, transport and service security concepts. For the code security the Java Code Security Concept [Gong 2003] is used. Within Openwings the code security begins with the installation of components where the signatures of the Java Archive[8] (JAR) files are validated and reaches into the processing of code where the access to different parts of the local platform (for example file system, JVM, network, etc.) is controlled.

[8]http://java.sun.com/j2se/1.5.0/docs/guide/jar/jar.html

The transport security focuses on the communication encryption between service objects by using the Connector Services.

The Security Service uses a role-based method access design. Which methods can be access by which roles is defined by a security policy (see Sub-section 7.4.3). Hence, it is possible to prevent unauthorized access to critical methods within a service object.

The topic of security in Openwings is discussed mored detailed in Section 7.5.

7.2.2 Contexts in Openwings

The context service offers the possibility to connect several platforms. Such connections can be used for three purposes, as described in the following.

At first, the platforms inside a context build a kind of trusted partnership. Service objects can be discovered unrestrained within a context by using discovery mechanisms. If a discovery manager is running outside a context it is not possible to find service objects from inside the context. This provides the system with a kind of border security where it is possible to define which service objects can be accessed from outside and which are not accessible.

Secondly, it is allowed to install and execute components not only on platforms but also on contexts. In this case, the context service decides on which platform a component is installed or executed. If a platform fails, components and running processes are tried to be moved to other well-running platforms.

And finally, it is possible to put policy objects in Context Services. This feature provides the possibility to assign a policy that affects all components installed on this context. Thus, it is easier to administrate a complete group of components.

7.2.3 Interfaces

Developers of components can specify different types of interfaces in Openwings using Java. The following types of interfaces are listed in the Interface Definition Specification [Bieber *et al.* 2003]:

- synchronous and asynchronous service interfaces,
- legacy system interfaces,
- data object interfaces,
- attributes interfaces and
- policy interfaces.

Synchronous and *asynchronous service interfaces* offer interfaces for the usage of the service behind and to connect to other services.

Legacy system interfaces are used by the connectors to make legacy systems in form of services available (for more details on connectors see Sub-section 7.4.4).

Data object interfaces control the access of objects, that are stored in a database.

By means of *attribute interfaces* it is possible to set parameters which are then used to discover particular services (see Sub-section 7.4.2).

Policy interfaces represent configuration data in Openwings and are stored in policy files. The information of a policy file is read by a policy object and can be accessed through the policy interface.

In the following sub-sections *synchronous* and *asynchronous service interfaces*, *attribute interfaces* and *policy interfaces* are described in more detail. These three interfaces are used in nearly every application.

Synchronous Service Interfaces

The synchronous service interface is used to specify methods which might be accessed from other services. Basically every Java interface can be used as an interface for Openwings components since Openwings does not provide a general interface from which the user interfaces are derived.

However, Openwings specifies some requirements for synchronous service interfaces and their methods. Thus, a valid synchronous service interface must fulfill the following conditions:

- Every method of a synchronous service interface must be able to throw the `RemoteException`. This forces the developer to deal with exceptions and to think about problems of distributed computing and error handling.
- Every used parameter must be serializeable, i.e. the parameter must be convertible into a bytestream. The reason for this is the established connection between service objects provided by Connector Services which limits the form of the transferred data to bytestreams. This applies also to return values.

The Openwings reference implementation currently supports Java Remote Method Invocation (Java RMI) and Internet Inter-ORB Protocol (IIOP[9]) as the synchronous and OpenJMS as the asynchronous communication protocol.

Asynchronous Service Interfaces

Openwings allows the specification of arbitrary asynchronous service interfaces for the communication between services.

The event producer uses the asynchronous service interface to generate events while the event listener implements the asynchronous service interface to be able to react on events which are generated with this interface.

Asynchronous service interfaces require the same conditions as the synchronous service interfaces but with one additional restriction. The methods must not have a return value.

[9]Information about the IIOP specification can be found on the website `http://www.omg.org`

Policy Interfaces

Policies are used to store and supply configuration data. Every policy has a policy interface to query the configuration data. Each policy interface defines a policy type. In Openwings some policy types are already defined like *InstallableComponentDescriptorPolicy*, *Provide-, Use-, Publish-, EventServicePolicy* or *Management Policies*.

The *InstallableComponentDescriptorPolicy* contains information for the installation of the component (see Sub-section 7.4.5 for details). In order to describe usable service interfaces the *ProvideServicePolicy*, *UseServicePolicy*, *PublishServicePolicy* and the *EventServicePolicy* can be used. *Management Policies* on the other hand, are used to provide configuration data for management services. To be able to read and write through policy interfaces it is needed to generate a sequence of classes. In Sub-section 7.4.3 the generation of such classes and the usage of policies is described in more detail.

Within this section, an overview over Openwings is given. In the following section Components in Openwings, which are used to develop software, are introduced.

7.3 Components in Openwings

Source code is handled by Openwings in form of components. A component contains the implemented software combined with supporting information which is needed by Openwings and is packed into a JAR file. Openwings distinguishes between executable and non executable components. The interfaces are provided in non executable components while the implementation of the interface is packed into an executable component. Therefore, two components are needed for one service (for example one non executable component A_i which contains the interface and one executable component A_e which includes the implementation) but it is possible to start more than one service from one component.

In the following section the usage and the parts of components in Openwings are introduced.

7.3.1 Parts of Openwings Components

As already mentioned in the previous section, components for installation are offered in form of JAR files in Openwings. During the installation, the files contained in the JAR file are installed on the local platform. In order to recognize the usage of the different files by the framework it is necessary to put the files in different directories which are referenced in the Installable Component Descriptors (see Subsection 7.4.5).

The Openwings tutorial [Openwings 2003] proposes the following directory structure:

- /bin
- /policies
- /data
- /docs
- /lib
- /http
- /source

The */bin* directory contains executable files which may be executed by the container service from the current operating system. Such files can be for example .bat or .csh files. Openwings provides a mechanism which automatically executes different files, regarding to the operating system. Another way to execute native code is offered by the Java API as described in [JNI 1997].

All files in the */policies* directory are recognized as policy files by the framework.

Files used by services may be located in the */data* directory. The name of these files must be specified through attributes (for more information about attributes see Sub-section 7.4.3) in order to be able to access the file.

The content of the */docs* directory will be published on a webserver. This directory should be used for textual descriptions and documentations of components (for example Javadoc[10] files) and should contain an `index.html` file.

The JAR files are located in the */lib* directory. The */lib* directory in general contains all Java and native libraries which are used by the services. All libraries in this directory can as well be used by services from other components.

Further, the */http* directory contains all files which should be published by the webserver.

If the code should be published with the component, it is possible to put the source code in the */source* directory.

7.3.2 The different Types of Components

As already mentioned at the beginning of Section 7.3, Openwings distinguishes between two different types of components, executable and non executable components.

Non executable components contain files which may be used by other components, for example service interfaces. Components which contain executable files are named *executable components*. An executable file can be a Java class or a script file which then may be executed by the operating system itself. The container service starts a process to execute a component and manages the process during its lifecycle.

[10] http://java.sun.com/j2se/javadoc/

With the help of executable components it is possible to create services which in turn publish their service interfaces. Executable components are divided, regarding to the type of their interfaces, into *provider*, *user*, *publisher* or *subscriber*. A *provider* offers synchronous service interfaces while the *user* uses such synchronous service interfaces. On the other side the *publisher* produces events with an asynchronous service interface and the *subscriber* is notified via asynchronous service interfaces about these events. In order to identify the purpose of a component, it is recommended to add these terms to the name of the component. Some cases do not allow to identify a component clearly. For example a component can be as *user* and a *provider* at the same time, or offer synchronous and asynchronous interfaces.

Within Openwings the components have different states. These states and the transitions between these states build together the lifecycle of a component. The lifecycle of components is described in the following section.

7.3.3 Lifecycle of Components

During its installation, a component runs through different states. Figure 7.4 shows the possible states of a component and the proper transitions between these states.

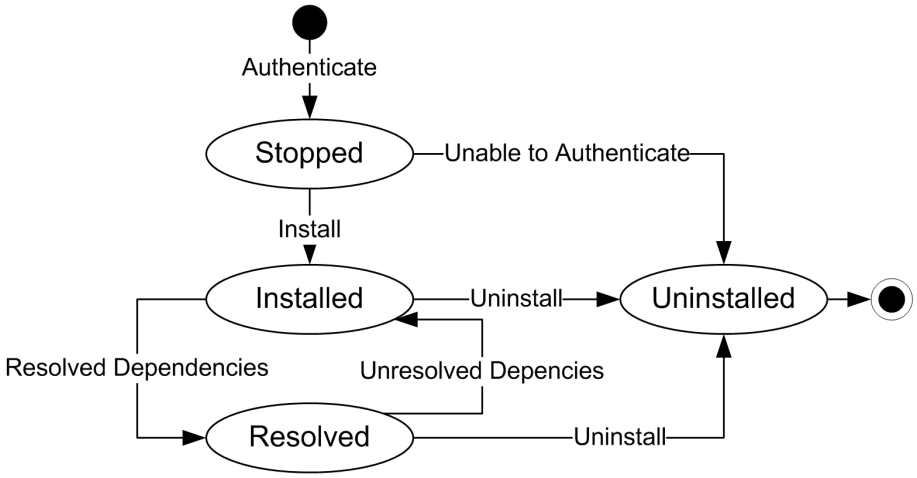

Figure 7.4: Component Install State Diagram [Bieber and Carpenter 2001]

The possible states of a component are *Stopped*, *Installed*, *Resolved* and *Uninstalled*. Additionally there are two other states where the component is not known by the Openwings framework. These two states are marked in Figure 7.4 as dots and can be seen as entry respective exit points. The transitions describe the possible changes of the component state and are depicted as arrows. There are seven transitions:

- **Authenticate:** The authentication transition starts from a state where the component is not known by the framework. At the beginning of the installation the signature of the JAR file is checked. This prevents of installing untrusted code on the platform.

- **Install:** After a successful authentication, the install service creates a new directory for the component. This directory is located in the installation directory of Openwings and has the same name as the component. The content of the components JAR file is extracted into this directory. After a successful installation the component's state switches to Installed.

- **Resolved, Unresolved Dependencies:** During these transitions the install service tries to replace the references and attributes of the Install Component Descriptor with the correct values. If it is possible to replace all references the component changes into the Resolved state. Otherwise, the component remains or switches back to the Installed state. If a component is not executable and the component is in the Resolved state, it can be used by other components whereas executable components are ready to be executed at this state.

- **Uninstall:** During the uninstallation of a component all files owned by this component are removed. If processes of the component are running or the component is referenced by other components and the component should be uninstalled, the component is not removed immediately but marked for later uninstallation. Further, it is important to mention, that in Openwings, components marked for uninstallation are removed during the next start of the install service.

If an already installed component is installed again with the same version number, name and unique ID, the old files are replaced by the new one. Services started from this component are not interrupted by such an installation. The data contained in the new files is loaded by the service when the files are accessed the next time.

7.3.4 Relations between Components

Openwings knows only one type of relation between components: Component A is only related to component B if component A uses the classpath, the codebase or a parameter of component B. Therefore the Installable Component Descriptor of component A contains a reference to component B. A component can only reach the resolved state (see Figure 7.4) if all components on which it depends are in the resolved state. For more information about references see Sub-section 7.4.5.

In Openwings, it is not possible to construct relations between two executable components. If for example the executable component C wants to use methods of component D, it needs the interface of executable component D. Therefore, each interface is offered in one separate component. With this concept it is easily possible to make a service available to many other services. The interface must only be published once and then all other services may use it.

As already mentioned, components provide the possibility to develop software and deploy it in Openwings. The process of software development and what has to be considered during the software development is discussed in the following section.

7.4 Software Development with Openwings

In this section, the offered concepts to implement and use components in Openwings are introduced. At first, the initializing of service objects which are created out of components is treated (see Sub-section 7.4.1), followed by the description of the lookup and usage process, the service parameters and the concept of service listeners (see Sub-section 7.4.2). Therefore, the underlying policy concept and the connectors are described (see Sub-section 7.4.3 and Sub-section 7.4.4). At last, the concept of Installable Component Descriptors is described (see Sub-section 7.4.5).

7.4.1 Creation and Intializiation of Service Objects

With the aid of services from the Openwings framework every software program is able to create and use service objects. This is done by using executable components. As already depicted in Sub-section 7.3.2, executable components consist of an executable Java class, which then creates the service object. In Openwings the container service starts the executable component.

Listing 7.1 shows how service objects are created and distributed. `ComponentComplex` is taken from the `ComponentFactory` and a new instance of the class `HelloWorldProvider` is generated. The instance of `HelloWorldProvider` is linked with the `ComponentComplex` and published by calling the method `provideService`.

```
public static void main(String[] args)
{
  // Get a simple component service from the component factory
  ComponentComplex component = ComponentFactory.getComponentComplex();

  // Create a new service provider.
  HelloWorldProvider helloWorldProvider = new HelloWorldProvider();

  ...

  // This exception block will catch any errors trying encountered in providing
  // the service
  try
  {
    // Distribute the HelloWorldServiceSynchronous interface on the
    // helloWorldProvider object so it can be remotely invoked.
    Object distributedObject = component.distributeObject(
      HelloWorldServiceSynchronous.class, helloWorldProvider);

    // Publish HelloWorldServiceSynchronous interface on the distributedObject.
    // Save the unique identifier of the object being published.
    UniqueID serviceID = component.provideService(
      HelloWorldServiceSynchronous.class, distributedObject);
  }
  catch (Exception e)
  { ... }
...
}
```

Listing 7.1: The main method of a Java class where a service object is created, distributed and published.

Openwings is further able to start the component automatically after the instal-

lation. If the component refers to more than one Installable Component Descriptor (see Sub-section 7.4.5 for details), the component will not be installed more than once, but started multiple times. As a result of this characteristic, it is possible to create several service objects with the same type but with different configurations.

After the creation of service objects, the initialization of them follows a pattern, which is the same in most of the applications and takes place in the constructor of the component class.

At first, a component service object is generated by using the `ComponentComplexFactory`. After that, the returned service object can be initialized.

Since the service object was already distributed by the code of the `main` method it is possible to use it as long as the `shutdown` method is not called.

The Openwings framework requires the implementation of a `shutdown` method to be able to delete a service object. The `shutdown` method enables efficient memory usage and helps the garbage collection releasing memory.

Finally, it hast to be mentioned that class specific initialization must be done within the constructor and therefore is not explicitly described in this section.

7.4.2 Usage of Service Objects

Service objects with a synchronous service interface can be used by the `useService` method. With the method `subscribeService` an event listener, which receives events from an asynchronous service interface, can be connected. The interfaces `Component` and `ComponentComplex` provide two possible ways of using a service object.

The first possibility is to enter only the service interface of the needed service object. Second, it is possible to specify parameters for the needed service object or the used service listener which limits the set of fitting service objects and listeners.

With service parameters it is possible to specify the attributes of a needed service object. This can take place within the call of the `useService` method by specifying the needed parameters. It is for example possible to use only service objects which are running on a specific platform by indicating it within the call of the `useService` method.

The service listener is a service object, which reacts when new service objects become public and available or are stopped. In such cases, the component service generates events which can be received by implementations of the service listener.

To be able to receive events, the service listener must implement the interface `UseServiceListener` for synchronous and the interface `EventServiceListener` for asynchronous service interfaces.

To register a service object as a service listener of another existing service object the methods `useService` or `subscribeService` are used (see Listings 7.2 and 7.3 below). The object which implements the service listener interface is used as a parameter for these method calls.

```
// use a service
try {
  component.useService ( ServiceSynchronousInterface.class,
      UseServiceListenerlistener )
}
catch ( InvalidServiceException e ) { ... }
// discard a service
discardUsedService ( serviceID )
```

Listing 7.2: A code snip where the component registers itself as the service listener for another service object with a synchronous service interface.

When an `EventServiceListener` is registered, an object, which is capable of receiving events as well as implements the asynchronous service interface is needed.

```
// subscribe to service
try {
  component.subscribeService ( ServiceAsynchronous.class, java.lang.Object
      service, EventServiceListenerlistener );
}
catch ( InvalidServiceException e ) { ... }
// unsubscribe from service
unsubscribeService ( someServiceAsynchronous.class )
```

Listing 7.3: A code snip where the component registers itself as the service listener for another service object with an asynchronous service interface.

Before service objects can be used it is necessary to configure the underlying components. Openwings provides a possibility where this configuration data is stored in external files and uses the policy concept described in the following section.

7.4.3 Policy Concept

One of the guidelines for the specification of Openwings was:

> *"The Openwings architecture focuses on zero administration, plug-n-operate (PLOP) systems."* [Openwings 2003]

To realize zero administration, Openwings utilizes the policy concept. In Openwings it is possible to store configuration information in so-called policy files where the data is arranged in XML format. In this sub-section the policy concept and its usage is described.

To be able to read and write policy files, a row of classes and files must be generated beforehand. Within the Openwings reference implementation [Bieber and Carpenter 2001], this task is assigned to the PolicyBuilder. The PolicyBuilder takes the interface of the policy and generates the following building blocks:

- A class which implements the given policy interface and is capable of reading and writing the policy files.
- An XML schema which defines the format of the XML file.
- A class which provides a command line tool for creating policy files.

In Figure 7.5 all parts of the policy concept used by Openwings and their relations among each other are shown. Only the "own policy" interface is needed to generate all required files by using the PolicyGenerator.

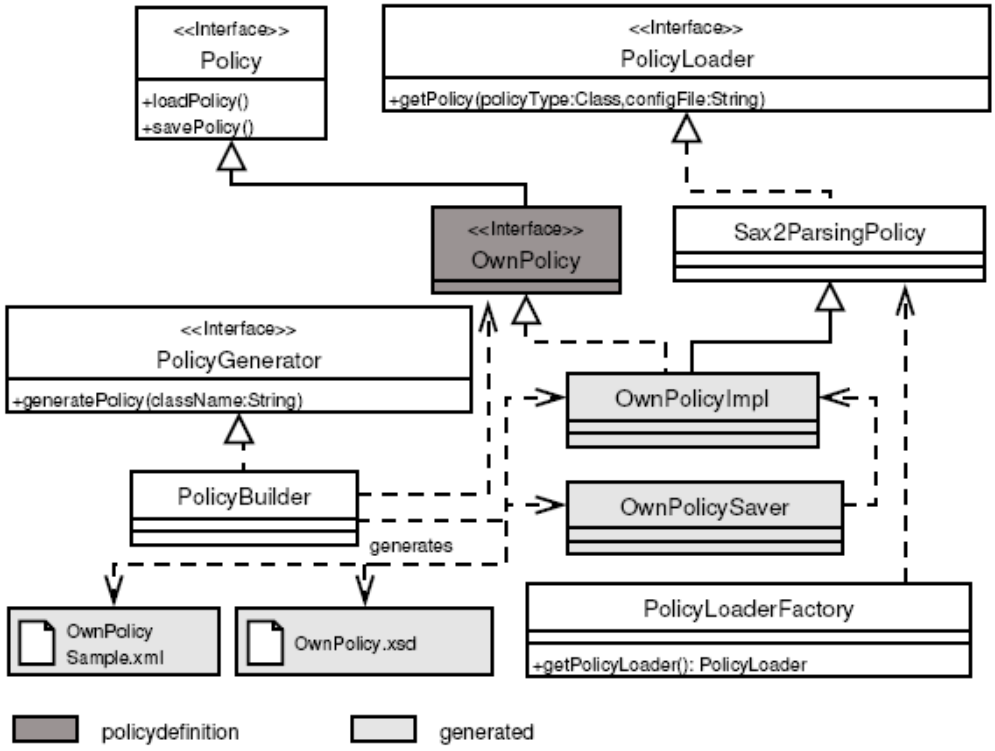

Figure 7.5: Openwings Policy Concept

To load policy data a `PolicyLoader` object is needed, which can be retrieved from the `PolicyLoaderFactory`. The `PolicyLoader` is used to load policy data only requiring the policy interface and the filename of the policy file as parameter. In return, the `PolicyLoader` generates a policy object. This policy object is then able to access the policy interface by reading and writing the policy data.

The disadvantage of such a policy concept is that policy objects are not reacting on changes of the underlying policy file. As a workaround for this drawback an extended PolicyGenerator may be useful which allows policy objects to monitor the policy file for changes.

7.4.4 Connectors

A connector enables the communication between service objects through service interfaces by using middleware. During the implementation of service objects the developer does not need to take care about the used middleware.

The Connector Service offers the possibility to generate connectors. A connector generator produces a connector for a specific service interface. The reference implementation of the Openwings specification [Bieber and Carpenter 2001] provides connector generators for RMI, IOOP and JMS connections. The connectors must be generated during the compilation and packing of the component.

To integrate legacy systems Openwings uses connectors to be able to communicate with them. For this purpose, firstly the interface for the legacy system must be specified. With this interface it is then possible to generate connectors depending on the needed middleware.

The connectors provided by Openwings have big advantages compared to the usage of common middleware. A synchronous connector can try to call the method of the service object several times before reporting a failure. This can prevent error when services are running on different platforms and the connection is temporarily unavailable. Asynchronous connectors are able to offer a buffer in which events can be stored. This takes the load from event listeners since they do not have to react on events immediately after their occurrence.

These two advantages can be always achieved by using adequate middleware and are not linked to connectors. But the usage of connector generators makes the development of components, which communicate with other service objects, easier.

7.4.5 Installable Component Descriptors

An Installable Component Descriptor (ICD) describes the different parts of a component, attributes for the identification of a component and information needed to execute a component.

As already mentioned in Section 7.3, a component is packed in form of a JAR file which must at least contain one file (`./policies/InstallableComponentDescriptorPolicy.xml`). This file contains the description of an ICD in XML format. [Bieber and Crumpton 2003]

Openwings provides a graphical editor allowing the user to change settings of the ICD in a simple way. This editor is able to save the ICD in the required XML format and to load an existing ICD.

The ICD contains several parameters and defines their values. To provide flexibility, it is possible to assign not only values but references to the parameters. References are resolved during the installation of a component and are used to provide platform-dependent values for the parameters.

Installable Component Descriptor Parameters

In this sub-section the different parameters of the ICD are described. The following itemization lists the available parameters of the ICD:

- Unique ID
- Component Name
- Version
- Icon
- Description
- Executable
- Executable String
- Platform
- Shared Container Hint
- Restart Hint
- Boot Process Hint
- Run Immediately Hint
- Serve Hint
- Mobile
- Classpath
- Resolvable Classpath
- Codebase
- Resolveable Codebase
- Properties
- Command Line Parameters
- bin, policies, data, docs, lib, http and source

The *Unique ID* is a 16 byte string which identifies a component unambiguously and can be generated automatically. The *Component Name* represents the name of the component which should be unique as well because the component name is used by the framework for identification purposes. *Version* is equal to the version number. The Openwings Explorer is able to show icons for each component. The *Icon* parameter is the place where the file of the icon is located. The *Description* is

a simple textual description of the component which should be in human readable format. The parameter *Executable* indicates whether the component is executable or not. Within the *Executable String* the executable class or the native file of an executable component is specified. The used Java Virtual Machine for executable components using an executable Java class or the operating system for native files is described in *Platform*. *Shared Container Hint* specifies if the started processes from this component should run in a container together with other components. If the *Restart Hint* is set to "true" and the process terminates in a irregular manner (for example with a "not catched" exception), it is started again immediately. When the *Boot Process Hint* is set to "true" the component is started right after the system start. The *Run Immediately Hint* decides if the component is started right after the installation of the component. When the *Serve Hint* is set, the component's JAR file will be installed on a http-server. The *Mobile* parameter indicates if a process can be transferred between two platforms. For an executable component the *Classpath* contains needed libraries for the execution. In case of non executable components the *Classpath* offers libraries to other components. The content of the *Resolvable Classpath* will be added to the classpath, but unlike the *Classpath* it may contain references (see Sub-section 7.4.5). Libraries listed within the *Codebase* are offered by a http-server. This allowes the usage of these libraries also on other platforms. In general, libraries which can be used by other components should be put in the *Codebase*. The content of the *Resolvable Codebase* is resolved and added to the *Codebase* by the install service and behaves similarly to the *Resolvable Classpath*. *Properties* represent (key,value)-pairs and are handed over to the JVM during a component's call. These *Properties* can then be used by the service object. Properties from non executable components can be referenced from other components. In the *Command Line Parameters* the parameters for the `main` method can be set. *bin*, *policies*, *data*, *docs*, *lib*, *http* and *source* refer to directories under the base directory (see Sub-section 7.3.1).

Using References

Openwings allows the usage of references in the parameters *Properties*, *Resolvable Codebase* and *Resolvable Classpath*. References can be used as variables like in a programming language but with the limitation of assigning a value only once. For further releases of Openwings, it is planned that the usage of references is changed into the direction of policies.

Components are installed on the local hard drive of a platform and thus, the absolute path changes depending on the computer, but the ICD needs absolute paths. For this purpose, references are used to specify parameters without knowing the absolute path. For example, the Resolvable Codebase parameter may contain entries like `${Communicator_im.libdir}` which refers to the libraries of the component with the name `Communicator_im`. It is also possible to refer to other components in the same way.

With `${@<method>}` it is possible to call methods to retrieve the value of a property (for example `date=${@java.util.Date.toString}`). In Openwings, such

methods are not able to use parameters and must return values of type `String`. Furthermore, the methods must be specified in the classpath.

Moreover, references can point to files or to properties of the framework. A complete list of all utilization possibilities can be found in [Openwings 2003].

7.4.6 Summarization

In this section the main concepts provided by Openwings to develop components were introduced. Beginning with the process of the installation an initialization of service objects within the Openwings framework, the usage of these service objects, where Openwings offers synchronous and asynchronous service interfaces, was described as well as the policy concept, which describes possibilities for providing configuration information. Finally, the connector concept, which is used by Openwings to abstract from different communication and connection technologies, was introduced.

In the following section the security topic is depicted. Security is an important point during the process of software development. Openwings offers facilities to develop a secure system.

7.5 Security in Openwings

Security is an important and well-solved issue in the architecture of Openwings. The trade-off between easy administration, level of security and flexibility is considered by Openwings to minimize administration while not reducing the level of security.

As already mentioned in Sub-section 7.2.1, Openwings focuses on code as well as on transport and service security. Security is handled in Openwings in a distributed way. There are for example, the Security Service and the Install Service, both dealing with the code security, while the Connectors are used to provide transport and service security. Other parts of the security concept are located in the Context Service and the Container Service.

In this section the provided security concept in general, which is divided into code security (see Sub-section 7.5.1), transport security (see Sub-section 7.5.2) and service security (see Sub-section 7.5.3), is discussed. Further the parts where they are processed are depicted.

7.5.1 Code Security

The code security concept used by Openwings is mainly based on the Java Code Security Concept [Gong 2003] and focuses on the avoidance of unallowed processes. In other words, Openwings has the possibilities to prevent the installation of malicious code and curtails the accessible parts on the platform.

Following the rules of the Java Security Concept, the Openwings code security is based on policies. In short, policies can be seen as configuration files and are stored in XML format (for more details see Sub-section 7.4.3).

The security policy consist of several grant clauses where every entry grants permissions to a specific component. With such grant clauses it is possible to control the rights of a particular component regarding platform features, and therefore restricting the influence of the code. For example component A is allowed to read from the file system (which is specified with grant clause in the security policy) but component B is not allowed to access the file system. Therefore, malicious code of component B has no possibility to access the file system since no permission is granted to it.

To prevent the installation of unwanted code, Openwings uses the installation service (see Sub-section 7.2.1) where the signature of a component installation file is compared to a valid key in the Openwings keystore. Hence, the component installation file must be previously signed with a valid key.

Java also supports the concept of checking signatures on component installation files containing code and assigning permissions based on these signatures. For this purpose, the policy concept is taken and enables the possibility to grant different permissions to code with different signatures.

7.5.2 Transport Security

Transport security concepts are used to secure the communication between components in general. In Openwings, the transport security provides a secure communication between components via secure connectors. The connector service (see Sub-section 7.2.1) is built with the ability to encrypt the data sent between endpoints. The reference implementation provides secure connectors based on the Java Generic Security Services (GSS) API. [GSS 2000]

The reference implementation by General Dynamics provides an easy way to include transport security since the connector generator for GSS over RMI is included in the reference implementation.

7.5.3 Service Security

The service security completes the security topic in Openwings. As already described in Sub-section 7.2.1, Openwings takes usage of a role concept with access rights to control the service security. Every process that produces a service object owns a role. The corresponding service object has the same role.

Within the security policy (see Sub-section 7.4.3), it is specified which method of the interface can be used with a specific role [Bieber and Thrash 2003].

The roles in the Openwings reference implemenation are managed by the security service which uses a separate Java keystore for that and can easily be configured through a user interface. Each platform stores its own roles and it is not possible to exchange roles between different platforms. If the same role is needed on more than one platform it has to be configured manually on each platform.

Every connector, which is involved into establishing a connection, authenticate to the security service on the local platform. The security service delegates the active role to them and pops up a login window if no role is activated.

Openwings does not distinguish between local and remote services. In order to ensure secure access to local services from other platforms the Context Service is needed. Contexts in Openwings enables the possibility to control the system formation, to build a system of systems and to configure the security. A context provides procedures to encapsulate a system from external access. Only specified services and methods can be used by remote services. Contexts in Openwings were described more detailed in Sub-section 7.2.2.

7.6 Conclusion

Openwings uses Java interfaces to specify its service interfaces. Compared to OSGi (see Sub-section 6.2.2 for details), Openwings additionally supports asynchronous service interfaces.

Apart from the service interfaces, Openwings also provides an easy configuration concept. The configuration information is accessed through interfaces, which are called policy interfaces. Policy interfaces enable the access to the policy data which is stored in files on the local platform. Openwings does not observe these files and changed information is not available for the process until the next access to the file. The generation of policies is not very easy to handle. Thus, it is not reasonable to generate policy interfaces for each component.

The compilation of Openwings components can cause some problems because the supplied Ant[11] scripts are not very flexible and must be adapted for own purposes.

Further, the installation of components and the creation of process objects do not provide dynamic references. References which are already resolved are assigned with new values or resolved again only upon the next installation.

The security concepts provided by Openwings, which are described in Section 7.5, seem to be sufficient for building a secure system, presupposing a trusted local environment. A trusted local environment is needed, since the security policies are stored on the local file system in form of plain text. This gives attackers an access point to the security policy and allows them to changed it. Regarding remote access, the context service is appropriate to secure the system from forbidden access. Contexts are not supported by the current version of the Openwings reference implementation. Hence, for the proposed user modeling system in Chapter 8, a workaround is needed. The solution takes usage of the service security concept by using secure connectors and Openwings roles combined with code security where installed code must be signed with a valid signature.

The container service monitors the running processes which represent a service. If a service throws an exception, the affected component is started again and continues its work. This self-healing quality can be used to provide a stable system.

Openwings provides concepts for an automatic installation of components and an automatic creation of service objects. These concepts are not very powerful. The development of a runtime component on the other hand, where startup and shutdown tasks are processed, is quite useful.

[11] Description of the Ant compilation tool available at `http://ant.apache.org`

The strength of Openwings lies in the support of several communication and discovery technologies. For example, discovery plugins can be developed and added to Openwings as well as connector generators to integrate further communication protocols.

This chapter provides the needed information about the used service-oriented framework. The developed solution approach of a user modeling system, which is introduced in the next chapter, uses Openwings as the underlying environment.

8. Design and Implementation of the Modeling System

8.1 Introduction

The findings of the previous chapters are used to design a multi-purpose modeling system. A versatile user modeling system should consist mainly of two parts, namely a user profile and a user model. Both together are needed to store the information about a user. Additional parts of a user modeling system must deal with topics, such as storage, privacy, security and communication.

The proposed user modeling system of this chapter is based on the requirements which are defined in Section 8.2. These requirements represent the functional constraints of the user modeling system and are used as a base for the design process. The first design step is the architectural design and the definition of the use cases (see Section 8.3).

Service-oriented software design (see Sub-section 6.2.3) gives no accurate answer to the questions regarding how big a service should be and how much functionality should actually be packed into one service. Therefore, two different solution approaches, namely a micro- and a macro-approach, were designed and implemented with the goal of comparing them against each other. The whole functionality of the two different solution approaches for the modeling system is the same, but the partition into services is different.

The functional design and the implementation is described for each solution approach. This is necessary since different components and services are required. The first approach, which is introduced is the macro-approach (see Section 8.4) followed by the micro-approach in Section 8.5. At the end of this chapter, the evaluation of the approaches is depicted (see Section 8.6) by starting with the user scenarios, evaluation criteria and evaluation setup (see Sub-section 8.6.1 - 8.6.3). Each solution approach is examined under the same user scenarios and criteria followed by a comparison of the evaluation results.

8.2 Software Requirements of the Modeling System

The user modeling system is to be used for storing and offering information about users. This information is needed to personalize e-learning systems.

In many personalization e-learning systems, the user model is seen as a part of the system and therefore the user model can only be used within this system. Every personalized system uses its own user model, and therefore, several models of the same user exist within different personalization systems. To centralize such a user model, a stand-alone user modeling system is needed which gives several systems access to the user information.

Although the user modeling system is developed as part of the AdeLE[1] research project, the required user modeling system must work as a single application. To satisfy the needs of different adaptive applications the user modeling system must offer possibilities to add specific modeling components.

The software requirements of this section cover information regarding the functional and non-functional requirements of the software. It does not specify how the requirements are to be implemented or how they are to be designed. Although examples may be given, these are only examples to help explaining a requirement and thus do not have to be followed.

8.2.1 Functional Requirements

This section describes the requirements of the user modeling application that are purely functional. They describe a feature that must be present or met in the final application.

Provide Access to Arbitrary Adaptive Applications

The access to the user modeling system must be available for different application systems and must not be limited to one specific application. This requires an interface specification which provides a versatile communication. Further, an extra interface for an external eye-tracking application must be available, as the proposed system is used within the AdeLE research project. However, this interface only needs to be capable of receiving pre-processed data from the eye-tracking system. User information queries are only answered over the common interface.

Store Received User Information

The application systems are responsible for delivering user information to user modeling system. The user modeling system on itself does not collect user information directly but must store, organize and process the received data. The user information must be organized in a component-based manner, where raw data is recorded in a user profile. Based on this user profile, user models exploit this information, interpret it and infer additional information about the user. Appropriate methods must be applied to ensure this functionality, as user models aim at the semantic enrichment of the raw data of user profiles.

Initialize, Update and Deliver User Profiles and User Models

The implemented user models must be initialized with the stored user information. If the user information changes, the affected user models must be updated automatically. A further task of the user modeling system is to deliver user information in form of user models. Thus, an application system must be able to query the user modeling system for a specific user model.

[1] `http://adele.fh-joanneum.at/`

Module-based Architecture

The architecture of the user modeling system must be module-based in order to ensure the flexibility, reusability and extensibility of the system. Thus, modules must be able to perform one specific task. For example, modules might be *Profiler*, *Modeler*, *Data Handler*, etc..

Provide a User Interface

For monitoring and maintenance purposes, the user modeling system must offer an own user interface (UI). The UI must provide all possible tasks which can be done with the user modeling system. Therefore, the UI must enable to view and change a particular user model. Further, the presentation of the UI must be adjustable. Possible presentation forms are for example a Java Graphical UI (GUI), XML or HTML.

8.2.2 Non-Functional Requirements

This section describes the requirements of the user modeling system that do not relate to functional aspects. They describe a concept that should be adhered to in the resulting system and in the development of the system.

General Software Requirements

Performance: The software should operate at a reasonable speed and with a reasonable response time to commands.

Operational Platform: The software should be independent of the underlying hardware and operating system.

Service-based Implementation: The implementation of the user modeling system should work on a service-based framework. However, this framework must not limit the functionality of the system.

Data Storage: The data should be primarily stored on the file system but the system must be designed to allow access to a database as well.

Privacy and Security: The privacy and security of user information must be ensured by the user modeling system.

External Requirements: The software should not require any specialized hardware to run. Any other software resources required for operation of the program should be distributed with the main program.

Documentation

A documentation of the implemented system must be produced in form of Javadoc generated HTML files. The documentation of the design process is covered in Section 8.4 and Section 8.5.

Develop with view to further development

The development of the application should facilitate future additions and modifications to the software.

Encourage code reuse from other sources

Reuse of code from freely available sources should be encouraged for the development of the system.

The requirements enlisted in this section describe the main guidelines for the software design. The software architecture is described in the followings section (see Section 8.3) and is valid for both solution proposals (micro- and macro-approach).

8.3 Architectural Design and Use Cases

The architectural design of the modeling system depicts the system by specifying the main components and describing their function and the interactions between them. To illustrate their interactions use cases are introduced in Sub-section 8.3.1.

As shown in Figure 8.1 there are basically three architectural levels. The lower level represent the *Openwings Framework* on which the *Modeling System* is erected. The next higher level describes the *Modeling System* itself, which is subdivided into several functional units. The highest level is used by applications or *External Systems* like adaptive systems or test environments. *External Systems* are not addressed in this software design but they do not need to be based on Openwings. The only restriction for the *External Systems* layer is the communication interface, which is provided by the *Modeling System* and must be implemented by them.

The most important layer in Figure 8.1 is the *Modeling System* layer and is divided into the following sub-layers:

- Data Storage
- Profiler
- Modeler
- Tools
- Manager

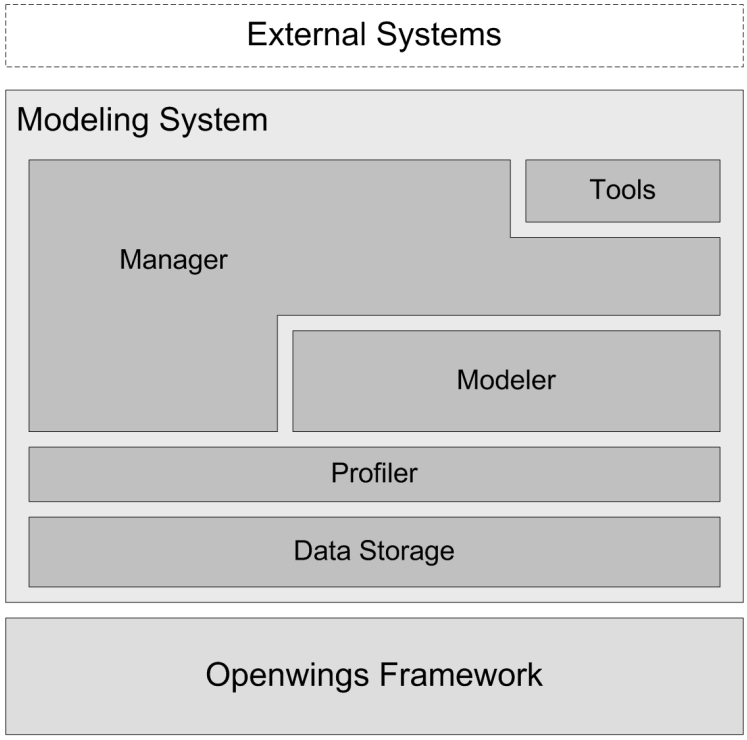

Figure 8.1: Layer Architecture of the Modeling System

The *Data Storage* layer represents the access to a data storage system like a database or the file-system. On top of the *Data Storage* is the *Profiler* layer. The *Profiler* must have a direct connection to the *Data Storage* since basically only user information in form of "raw data" is handled by the *Profiler*. The *Profiler* is accessed by two layers. First the *Modeler* layer, where the user models are hold and second, the *Manager* layer. The user models are initialized and updated with the data managed by the *Profiler*. The *Modeler* has no direct access to the *Data Storage* which prevents a concurrent manipulation of the data between the *Profiler* and the *Modeler*. If an application needs raw data about a particular user the *Manager* is able to access the *Profiler* and deliver the queried data. The *Manager* has access to the *Modeler* and to the *Tools*, since the *Manager* is responsible for offering *External Systems* access to the *Modeler*. The *Tools* layer provides the *Manager* with additional functionality and includes for example the GUI.

For external systems, the Modeling System behaves as one unit. This is shown in Figure 8.1 by the surrounding box of the different levels of the Modeling System. The communication with the *Modeling System* is processed by the *Manager*. Therefore, security policies, privacy methods and data encryption must be provided by the *Manager*.

So far, the main layers of the Modeling System and their tasks are described. In the following section, the use cases for the modeling system are introduced.

8.3.1 Use Cases for the Modeling System

Before the use cases can be described and analyzed, it is necessary to define the possible users of the modeling system. Considering the software requirements in Sub-section 8.2.1 two types of using systems can be identified within the context of the AdeLE research project, namely adaptive systems and one eye-tracking system. These two types of using or application systems have different purposes and therefore different use cases. Since the affected components of the modeling system are mainly the Modeler and the Profiler, it is also feasible to separate the use cases concerning the affected components. Figure 8.2 shows those use cases, which affect the Profiler and are initiated from an adaptive system.

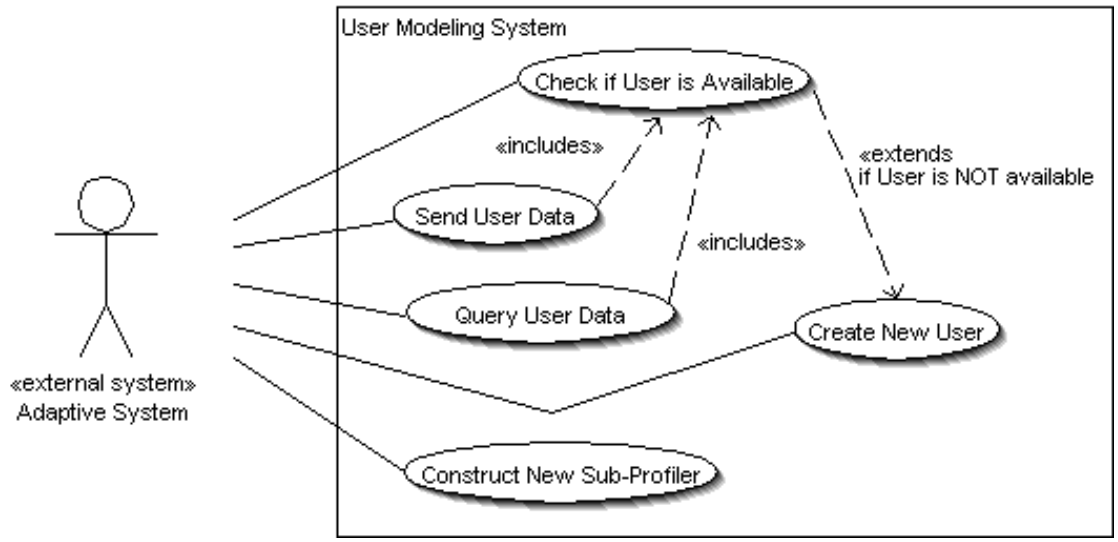

Figure 8.2: Use Cases for the Profiler

The actions of an adaptive system regarding the Profiler are basically *Send* and *Query User Data*. Further use cases are *Check if a User is Available*, *Create a New User* and *Construct a new Profiler*. The query for the user availability checks if a particular user is known by the modeling system. If this user it not known he can be added to the modeling system by performing the use case *Create New User*. In this case, the user profile is created and initialized. Finally, it is possible to add required profilers by external systems (*Construct New Sub-Profiler*). By this use case the external system sends information about the construction of the new sub-profiler. This information is used and a new sub-profiler is installed by the Profiler. The adding of sub-profilers works dynamically and allows to define arbitrary sub-profilers while the Profiler knows about the installed sub-profiles.

The use cases for the Modeler are similar to the use cases of the Profiler and are shown in Appendix A.1.

The second application system is the eye-tracking system. The eye-tracking system sends information about gaze movements of the user to the Modeling system (see Figure 8.3). This information is needed for later processing by the Modeler.

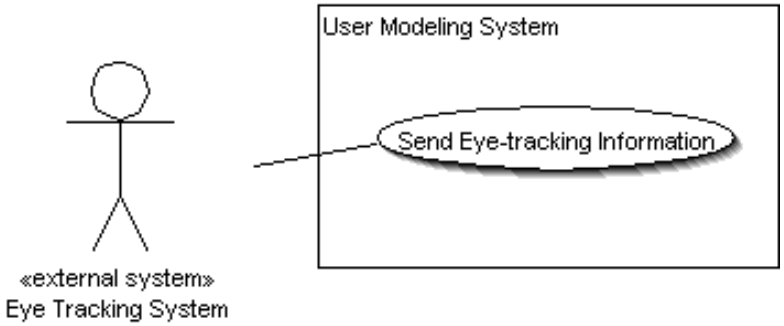

Figure 8.3: Use Cases for the Eye-tracker

The eye-tracker *Sends Eye-Tracking Information* to the modeling system where the Profiler records this data. This use case is very simple but it has to be considered that the amount of data received from the eye-tracker is very high and dense. Thus, the modeling system must be able to process this data within an appropriate time period.

The architectural design is on a rather coarse grained level of detail but it identifies the main levels of the modeling system and their interactions. In the following two Sections 8.4 and 8.5) the two proposed solutions (macro- and micro-approach) are described in more detail.

8.4 Macro-approach

After the architectural design process, the whole functionality of the system must be divided into functional units, according to the proposed architecture.

According to the guide-line of the macro-approach, the functional components are constructed by using the layers of the architecture, namely *Data Storage*, *Profiler*, *Modeler*, *Manager* and *Tools*. The resulting functional division is described in the following Sub-section 8.4.1. After the software design, the modeled system is implemented. Sub-section 8.4.2 depicts the implemented aspects of the macro-approach within the scope of the thesis project.

8.4.1 Functional Components

Considering the architectural design of Section 8.3 the determined layers of the architecture are used to form the components of the macro-approach. As shown in Figure 8.4, the following main components are identified:

- Data Handler
- Profiler
- Modeler

- Manager
- Profiler Editor

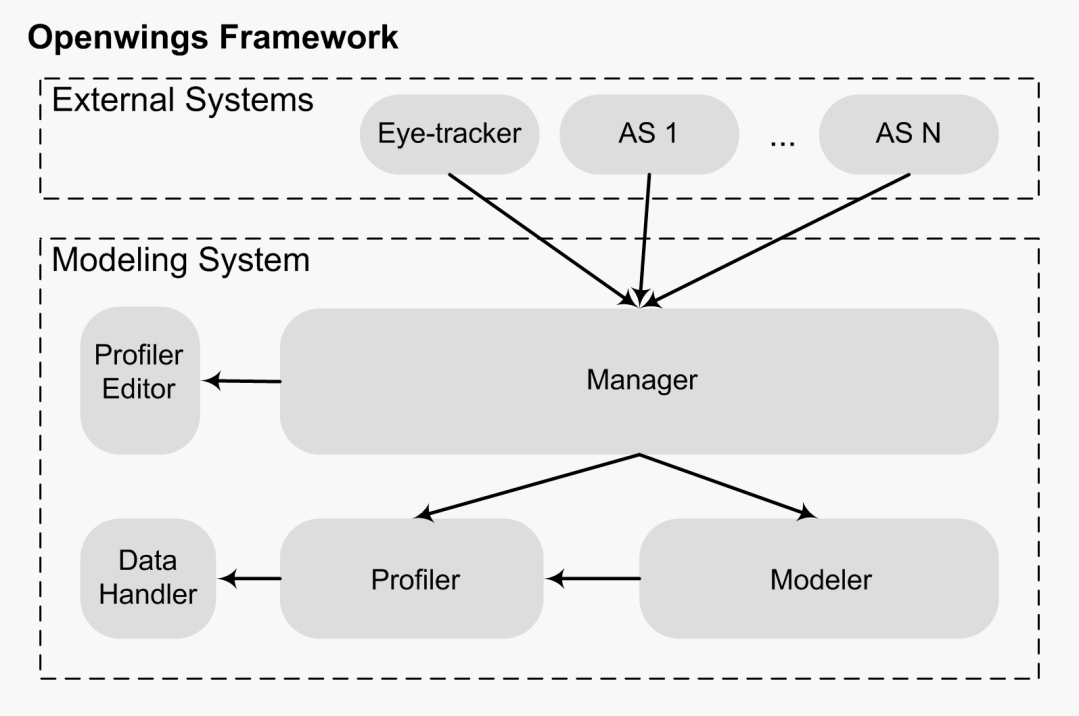

Figure 8.4: Functional Architecture of the Macro-approach

The *Data Handler* provides access to data storages (see Figure 8.4). There are no restrictions which data storage system is currently used. For example, it is possible to use a database or the file-system. This allows to simplify the usage of different storage system since all components which use the *Data Handler* do not need to know anything about the storage.

The *Profiler* performs all tasks which are needed to record, query and deliver a user profile. It is the only component which has direct access to the *Data Handler*. This access is used to fill the user profiles with recorded raw data about the users. There is a connection between the *Manager* and the *Profiler*, which allows external systems to query directly the user profile, because often a user model is not needed. There is a strict separation between the user profile and the user model based on the findings of Section 2.2. This separation is also realized in the components of the functional architecture. Further, the *Profiler* is based on a modular concept, which means that the complete *Profiler* consists of several sub-profilers, where each of the sub-profilers is responsible for a particular part of the user profile. This modularity allows to extend and adapt the *Profiler* according to the needs of the external systems. Additional sub-profilers are defined by external systems in form of a request to install a sub-profiler. The installation of this sub-profilers is performed

by the *Profiler* itself. Further, the *Profiler* handles this sub-profiler and the access to them.

Similar to the *Profiler*, the *Modeler* is also designed in a modular manner. The *Modeler* encapsulates several user models in form of sub-modelers, each of them covering a specific user model. The *Modeler* itself knows about the implemented sub-modelers. User models are defined by external systems. The definition of the user model is included in the request to install a further sub-modeler. The *Modeler* is responsible to handle this request and manage the installed sub-modelers. This functional architecture of the *Profiler* and the *Modeler* allows a maximum of scalability to the requirements of external systems.

The *Manager* is mainly responsible for the communication with the modeling system. To allow communication, an appropriate interface is specified by the *Manager* and has to be implemented by external systems. Further, the communication is performed by using a defined protocol. The protocol must be versatile to enable the needed communication. Further tasks of the *Manager* are defined as follows:

- resolve and delegate commands,
- communicate with the *Profiler* and the *Modeler*,
- offer privacy and security techniques, as well as
- identify and authorize external systems.

If a command is received by the *Manager*, it must resolve this command, extract the information and delegate this command to the resolved component (e.g. *Profiler* or *Modeler*). The affected component performs the required task and returns the result to the *Manager*. The *Manager* generates the return command according to the protocol specifications and sends this command back to the external system. Privacy and security techniques are also offered by the *Manager*. Security for the transfered user information is established by applying an encryption mechanism between the *Manager* and the external systems. To enable privacy, two concepts are used. First, by using pseudonymity (see Sub-section 6.3.1), the user is not directly connected to the information stored in the profiles and models. Second, the implemented user profiles and user models contain information about which part of the information is available for all systems and e.g. which part can only be read by defined systems. This concept of assigned level of privacy to user information is similar to the concept used in PAPI (see Section 4.4). As already stated, the user information is arranged in sub-profiles and user sub-models. This granularity is taken as the fundament for partitioning of the user information in order to assign the corresponding privacy levels. For example, the sub-profile "demographic data" could be assigned with the privacy rule that only a specific system A has read and write access. For all other systems this sub-profile is not accessible. The identification of the external systems is also done by the *Manager*, where each external system has its own *unique id*. This *unique id* is generated by the *Manager* during the first communication with the external system. Only the *Manager* and the external system have knowledge about this *unique id*.

The *Profiler Editor* allows to view and change the user profiles. Since the *Profiler Editor* is an internal system no privacy or security issues must be concerned. Further, it is possible to select the desired presentation form (for example, XML or HTML). The *Profiler Editor* is used to control and correct stored user information and to monitor the system activities.

External systems are mainly, but not restricted to, *Adaptive Systems*. The modeling system can be used by more than one *Adaptive System*, this is shown in Figure 8.4 through *AS 1 - AS N*. Additionally, the *Eye-tracker* is shown as an external system. The modeling system is able to process data received from an eye-tracking system. This data represents information related to gaze movements of the user and is included in the user profile and handled through the user model. To simplify the communication, the eye-tracking system communicates with the manager using the same interface as the adaptive systems. This centralizes the communication and centralizes the security concerns to the manager. The drawback is that the connected systems may influence each other in cases of access and processing time. Appendix B.2 shows a screen shot of AdeLE during a course. Information about adaptation is shown by offering alternatives to the selected presentation form (Instructional Alternatives) combined with relevant information from the user model (Profile from learner). To provide scrutable user models (see Section 5.4.2) a link to the particular user model is shown. This allows the learner to view and adjust his user model.

To illustrate the collaboration of the described components, Figure 8.5 shows the sequence diagram for the use case *Send User Data*. The *Adaptive System* calls the method `processCommand` of the *Manager*. At first, the security and the privacy policies are considered. This leads either into an authorization or into a refusion of the received command. If the authorization is granted, the *Context Manager* is invoked. The *Context Manager* resolves the required components (i.e. the context) to which the command is delegated based on the given command. The *Security* and the *Context Manager* are included in the *Manager* component but extra drawn to facilitate the explanations. The *Profiler* is called from the *Manager* to record the user data. The *Profiler* processes the given user data and stores it with the usage of the *Data Handler*. Additional sequence diagrams are shown in Appendix A.2 and A.3.

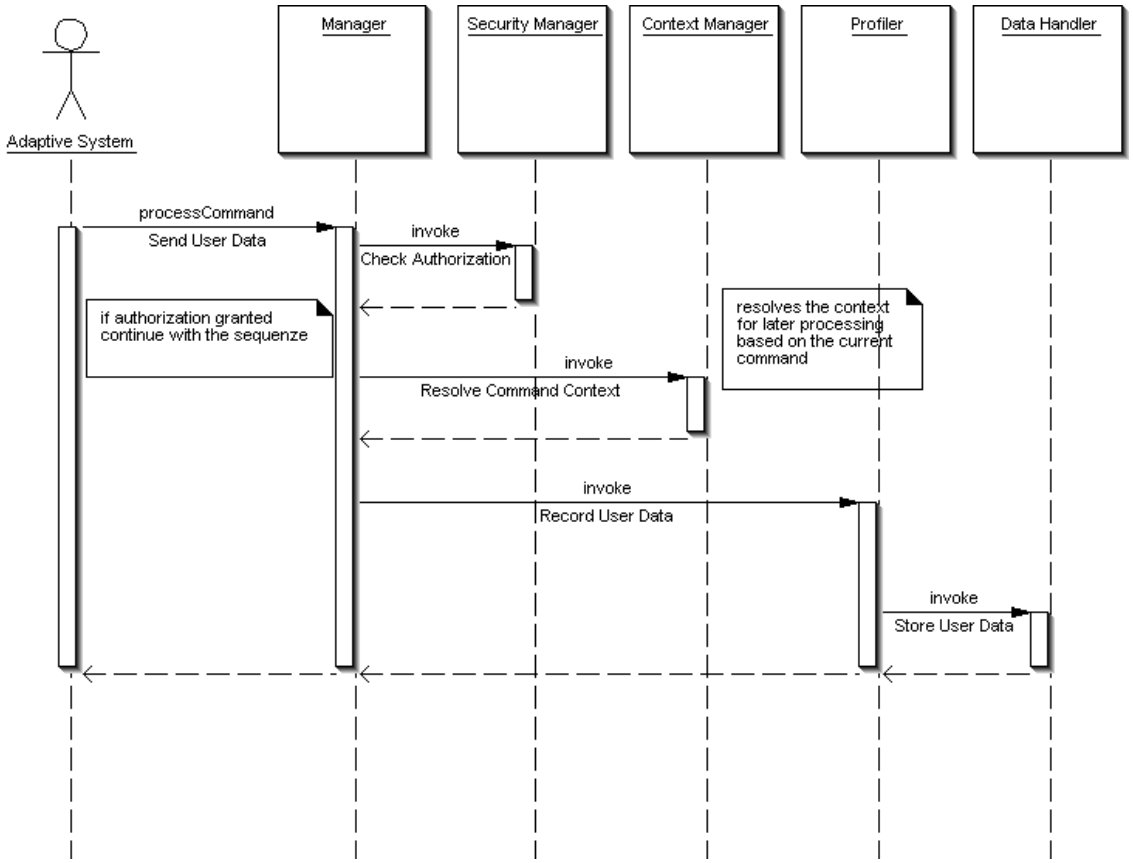

Figure 8.5: Sequence Diagram for the User Case *Send User Data*

The architectural design of Section 8.3 and the functional design of this section describe and specify the components of the proposed user modeling system. Implementations done within the thesis project are based on this design and are depicted in the following section.

8.4.2 Implementation of selected Aspects

The software design of the macro-approach is used to form the components of the first solution system by using the Openwings framework (see Chapter 7). In Openwings one component is packed into one service but a component can consist of more than one Java class. Openwings is not limited to Java but recommends the usage of Java to implement components and create services. The following Java components were implemented in the macro-approach:

- Manager
- Modeler
- Profiler

- Data Handler
- GUI Visualization

The Manager component consists of the classes `Communicator`, `CommunicationInterpreter` and `ContextManager` (see Figure 8.6). The `CommunicationInterpreter` uses the abstract class `CommunicationObject`. In order to be able to interpret the received commands, the `CommunicationObject` must be implemented. There are two implementations of the `CommunicationObject` available, namely the `EyeTrackerCommunicatioObject` and the `AdaptionSystemCommunicationObject` (see Appendix A.4).

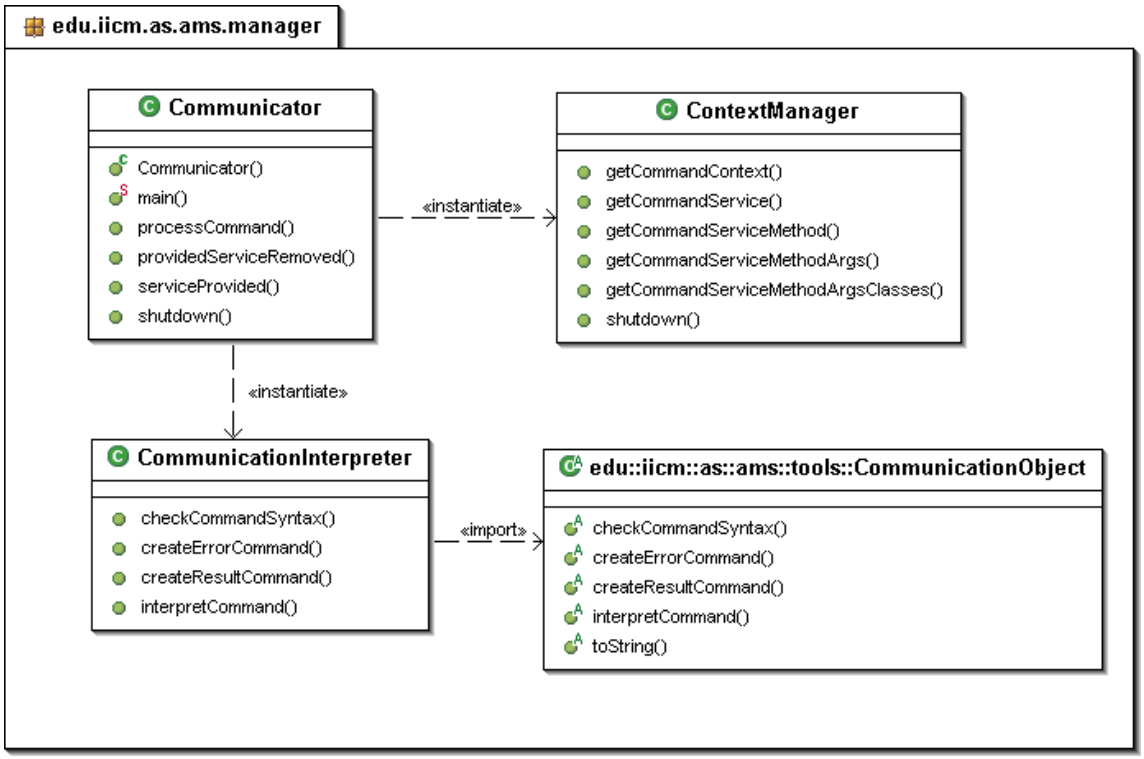

Figure 8.6: Class Diagram of the Manager Component

The classes shown in Figure 8.6 are conform to the functional description of Sub-section 8.4.1 except that there is no explicit functionality to enable privacy and security. Privacy is ensured by using pseudonyms whereas security is provided by using Openwings specific security features (see Section 7.5). Further improvements of the system must include security and privacy concepts, which are described in Section 6.3 and Sub-section 8.4.1.

The `Modeler` component is constructed to manage the `StateModeler` and the `WAVIModeler` (see Appendix A.5). Compared to the functional design of the previous section, this implementation does not allow to dynamically add user models. The implementation is limited onto these two user models, while the dynamic extension feature would allow to add user models during work. As shown in Appendix A.5,

each modeler encapsulates one specific user model, namely the `UserStateModel` and the `WAVIModel`.

Similar to the `Modeler` is the `Profiler` component (see Appendix A.6). The `Profiler` is divided into the `UserInformationHandler` and the `BehaviourHandler`. The `UserInformationHandler` is responsible for information about the user, as for example first name, last name, etc., while the `BehaviourHandler` records and manages data about shown instruction (`InstructionHandler`), interactions with the system (`ActionHandler`) and data from the eye-tracker system (`GazeTrackingHandler`).

Connection to a data storage system is provided by the `DataHandler` component (see Appendix A.7). The implementation of the macro-approach covers access to the file-systems (`FileHandler`) in form of XML documents (`XMLFileHandler`).

The GUI Visualization is a simplified form of the Profiler Editor, which is designed to allow different visualization forms. According to the requirements specified within the scope of this thesis, the Profiler Editor needs only a Java GUI (a screen shot can be seen in Appendix B.1).

As described in this section, the macro-approach consists of four components plus the GUI visualization which are packed into separate services. The implementation itself is based on the Java programing language where Java interfaces are used to define service contracts. For more details about how software is developed with Openwings see Section 7.4. The description of the implementation work in all details will exceed the scope of this thesis.

The first of two implementation steps was the macro-approach. Only four services are used within the macro-approach to provide the functionality of the modeling system. The second step is the micro-approach, where it is tried to pack as little functionality as possible and reasonable into one service.

8.5 Micro-approach

This section describes the functional design and the implementation of the micro-approach. As already stated in Section 8.1, the micro-approach follows the guideline that every component is represented by one service. Nevertheless, the functionality of the micro-approach is equally to that of the macro-approach (see Section 8.4). The design of the micro-approach results in a more fine grained structure of components and services.

At the first state of the functional design, the functional components have to be determined. This is done in Sub-section 8.5.1. The components as the outcome of this design are then used for developong the services. At the end of this section the implemented aspects of the micro-approach are depicted (see Sub-section 8.5.2).

8.5.1 Functional Components

The functional components of the micro-approach are created by sub-dividing the components of the macro-approach (see Sub-section 8.4.1). This is a simple way to

ensure the same functional scope as the macro-approach. The functional components are described within this section by a similar grouping as in the macro-approach (see Figure 8.4)

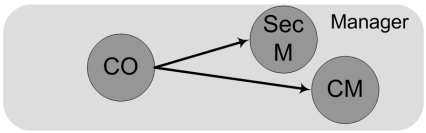

Figure 8.7: Functional Architecture of the Manager

The first group of functional components is the Manager (see Figure 8.7). The Manager is sub-divided into the *Communicator* (CO), the *Security Manager* (SecM) and the *Context Manager* (CM). The *Communicator* is responsible for the communication between external systems and the modeling system. The *Communicator* asks the *Security Manager* if the external system is authorized to access the modeling system. If access is granted, the relevant component for the current command is resolved by the *Context Manager*. For example, if the command contains user data which is to be stored in the user profile, the *Context Manager* tells the *Communicator* to proceed with the process by calling the Profiler.

The Modeler group (see Figure 8.8) contains functional components which are responsible for the user modeling. The access point to the Modeler is the *Model Manager* (MM). The *Model Manager* is responsible for the communication with the Manager. Further tasks of the *Model Manager* are to construct, initialize, deliver and maintain user models. Every user model is managed by its own Modeler. Examples for implemented modelers are the *State Modeler* (SM) and the *WAVI Modeler* (WM). The *State Modeler* contains the state model, which models the current user state regarding domain knowledge. A knowledge state (like "read" or "learned") is assigned to every piece of the learning material. For example, the knowledge state of user X about chapter 9 of course A is "learned". The *WAVI Modeler* represents the WAVI model, which describes the learning styles of a particular user. General information about the content of a user model in adaptive e-learning systems is described in Sub-section 3.4.1.

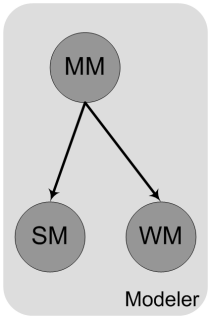

Figure 8.8: Functional Architecture of the Modeler

Raw data about the user is hold in a user profile. The user profile is sub-divided into several Profilers which are managed by the *Profile Manager* (PM) (see Figure 8.9). Similar to the Modeler, the *Profile Manager* creates and handles these Profilers. Implemented initial Profilers are the *User Information Handler* (UH) and the *Behavior Handler* (BH). The *User Information Handler* deals with general information about the user, as for example demographic data. The *Behavior Handler* holds information about the user behavior. The user behavior is observed by recording shown instructions (*Instruction Handler* IH), user interactions (*Action Handler* AH) with the system and gaze information (*Gaze Information Handler* GH). Gaze information is received from the eye-tracking system.

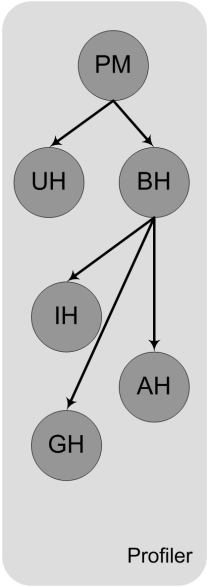

Figure 8.9: Functional Architecture of the Profiler

The last group of functional components is called Tools. The Tools group combines all components which are not directly connected to any other previously described groups (see Figure 8.10). These are "common" components used by the Manager and by the Profiler. The Tools group provides functionality to the Manager in form of the *Profile Editor* (PE) and the *Communication Interpreter* (CI). The *Profile Editor* offers a user interface to view and modify the user information stored in the modeling system. An implementation of the *Profiler Editor* is the *GUI Visualization* (GV) which provides a GUI for the modeling system and therefore offers an ergonomic way to view and change the user information. Further implementations of the *Profiler Editor* may include for example an HTML presentation. Since the communication with the *Communicator* is based on a specified command syntax it is useful to model an extra *Communication Interpreter* which takes the received command, checks the syntax of the command, interprets it and sends the result back to the Communicator for further processing. Thus, the *Communicator* is independent of the used command syntax. For each command syntax, a corresponding *Communication Interpreter* must be implemented. Examples are

the *Adaptive Systems Communication Interpreter* (AI) and the *Eye-tracking System Communication Interpreter* (EI).

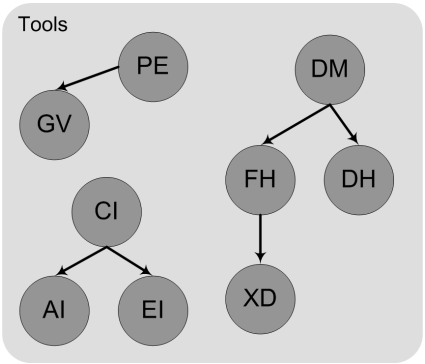

Figure 8.10: Functional Architecture of the Tools Component

The Profiler uses the Tools group to get access to a data storage. The *Data Manager* (DM) provides access to several storage systems. This makes the Profiler independent of the used data storage. Each data storage system requires an implementation of the *Data Manager*. Database access is provided by the *Database Handler* (DH), wile access to the file-system is realized by the *File Handler* (FH). The data organization of the *File Handler* is specified by an *XML Data Handler* (XD).

Figure 8.11 shows the complete functional architecture of the micro-approach. The previously described component groups are combined and the connections between components are depicted. External systems within the scope of the AdeLE project are represented by an *Adaptive System* (AS), an *Eye-tracking Systems* (ET) and a *Test Client* (TC).

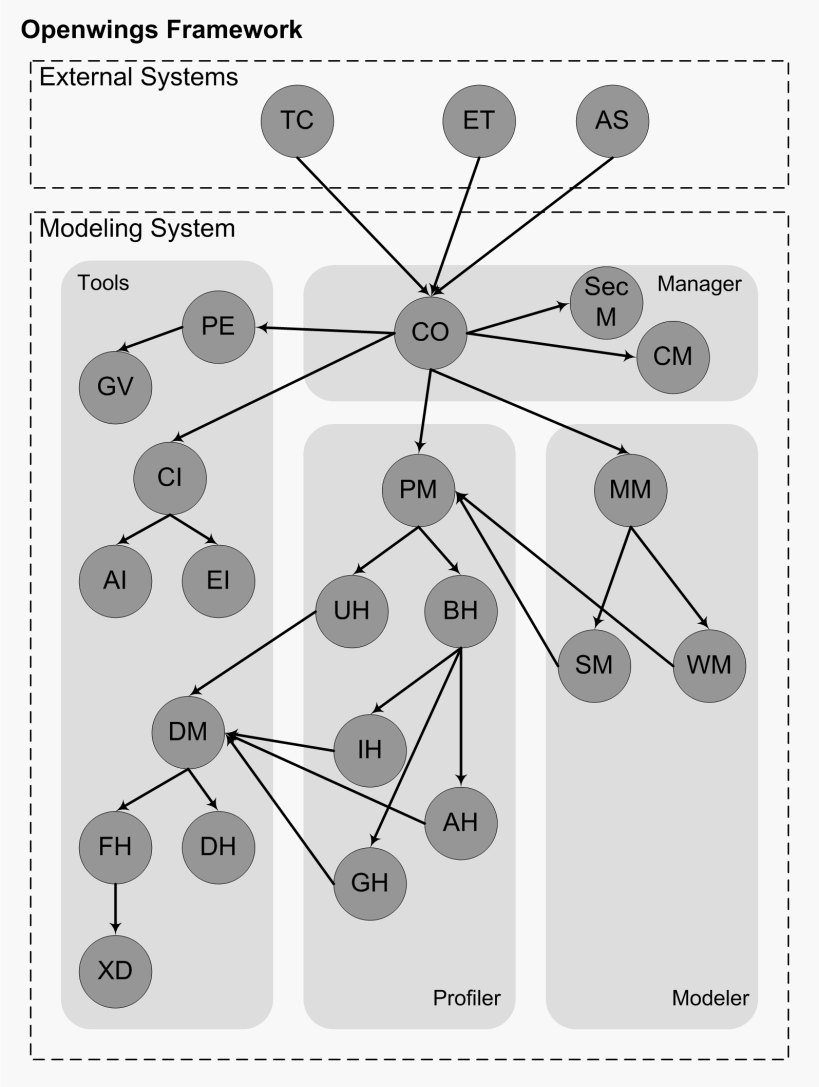

Figure 8.11: Functional Architecture of the Micro-Approach (adapted from [Gütl and Garcia-Barrios 2005])

Based on the architectural design (see Section 8.3), the functional design specifies components and their relations among each other. The following section describes the implementation of these components, which was done within the thesis project.

8.5.2 Implementation of selected Aspects

After the implementation of the macro-approach it was necessary to implement the micro-approach with the identically functionality as the macro-approach. Openwings allows to specify service contracts in form of Java interfaces. This ability is used to create a service for each class, whereby the existing classes from the macro-approach were re-used and additional interfaces were specified for them. This allowed

to reduce the implementation work for the micro-approach and guaranteed the same functionality for both approaches.

The manager component from the macro-approach is sub-divided into two services, namely the `ContextManager` and the `CommunicationInterpreter` as shown in Figure 8.12. The interfaces shown in Figure 8.12 are the service contracts. Naming conventions, like adding `Synchronous` to the end of the interfaces indicate that this service contract enables synchronous communication with the services. The synchronous communication is based in RMI (see Sub-section 7.4.4).

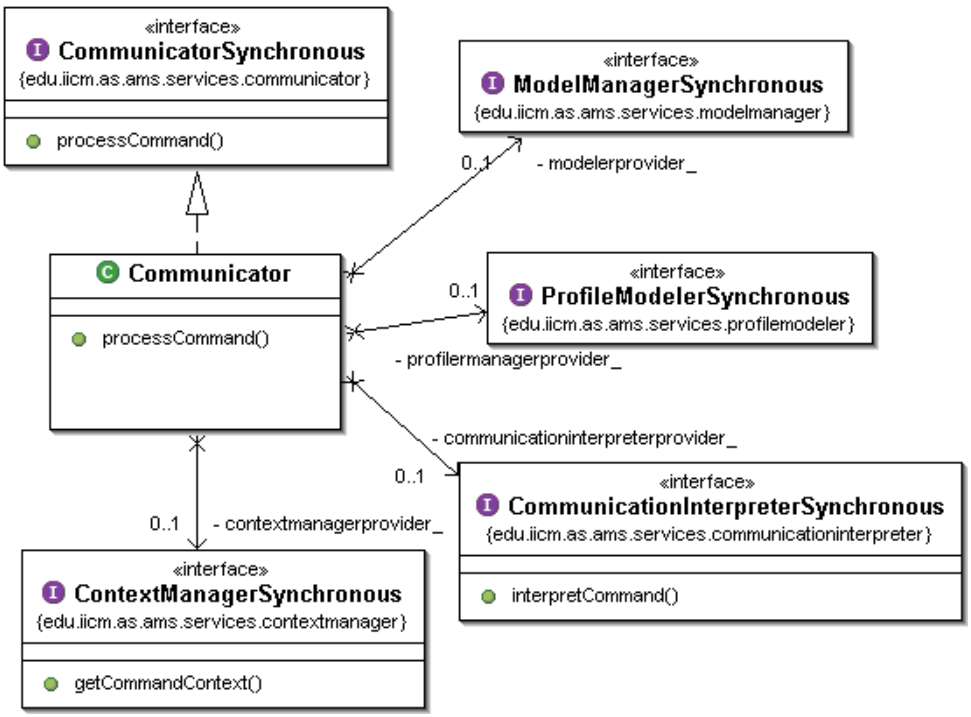

Figure 8.12: UML class diagram including services which are connected to the *Communicator* service

The `Communicator` service further connects to the `ModelManager` and to the `ProfileModeler`. The services of the Modeler group are `ModelManager`, `WAVIModeler` and `StateModeler` (see Appendix A.8). The `ModelManager` service is responsible for the implemented user models. As in the macro-approach (see Sub-section 8.4.2), two user models, namely the WAVI model and the state model are implemented. For each of these user models a modeler service is created. Access to user information, which is needed to initialize and update the user models is provided by the `ProfileModeler` service.

The functional architecture of the micro-approach (see Sub-section 8.5.1) proposes the partition of the Profiler into a `ProfileModeler` and several sub-profilers. Sub-profilers are allowed to have again their own sub-profilers. This results into the implementation of the Profiler as shown in Appendix A.9. The `ProfileModeler` uses two sub-profiler services, namely the `UserInformationHandler` service and the `BehaviourHandler` service. The `BehaviourHandler` has again sub-services, which

are called `ActionHandler`, `InstructionHandler` and `GazeInformationHandler`. All these profilers have access to the data storage.

The `FileHandler` service (see Appendix A.10) provides general access to the local file system. An implementation of the `FileHandler` is the `XMLFileHandler`, which organizes the data in form of XML files. Listing 8.1 shows an XML code-snip of the user information file.

```
<?xml version="1.0" encoding="UTF-8"?>
<profile>
  <firstname>Christoph Johann Felix</firstname>
  <lastname>Froeschl</lastname>
  ...
</profile>
```

Listing 8.1: XML user information stored on the file system.

In sum, the micro-approach contains 15 services. That is much more than in the macro-approach. Since 15 services are difficult to handle especially during the start and the shut-down of the system, it is useful to develop a runtime environment which keeps track of and monitors the life cycle of the services.

Not only handling but several other criteria are used to evaluate the two solution approaches in the following section.

8.6 Evaluation of the Approaches

An evaluation of both solution approaches introduced in Section 8.4 and 8.5 is necessary to find advantages and drawbacks of each approach. Further, the foundation should be established to be able to answer the question, how much functionality can be reasonable packed into one service. The evaluation follows the guidelines of a scenario-based evaluation, where stakeholders are used in form of scenarios.

The following section is not a complete evaluation but a first step to classify the solution approaches in a technical way. This evaluation is made in four steps. First, different user scenarios which are relevant for the evaluation are described in Sub-section 8.6.1. The second step is the definition of the evaluation criteria or benchmarks (see Sub-section 8.6.2), followed by the third step, the description of the evaluation setup in Sub-section 8.6.3. Finally, the results of the evaluation are depicted in Sub-section 8.6.4.

8.6.1 User Scenarios

The evaluation of the approaches is performed under the conditions of these user scenarios. User scenarios are often used for user-centered design in the field of human-computer-interaction (HCI) [Bødker 2000]. Basically, the user scenarios used in this section differ from their common meaning in that these scenarios do not describe hypothesized user interaction scenarios but roles of the users. Each role has its own goals and tasks to be carried out. Thus, different user scenarios are affected by the two approaches in different ways. The term user scenarios is applicable

because they are used to compare different solutions, which is similar to a HCI evaluation.

The following section describes three user scenarios, namely *Administrator*, *Developer* and *Application User* by depicting their motivation and possible interactions with the modeling system.

Developer

The Developer scenario includes persons which are involved in the design and implementation process. For example, these persons are software developers, software designers and programmers.

Main goals of Developers are to create a stable, useful, powerful and versatile modeling system. Developers are compelled to fulfill the given requirements of Section 8.2. Further, Developers intend to spent as less time to develop the system as possible. During testing, they have to deal intensively with the system by simulating real-world tasks. Since Developers are basically human beings, another issue is the survey over the source code during the design and implementation period.

Application User

Application User is the scenario for which the modeling system is mainly developed. Application Users of the modeling system are adaptive systems, eye-tracking systems and any other system with the need for user information.

Goals of Application Users are to store and retrieve user information in an easy, fast, robust and secure way. There are no interests in what is happening behind the communication interface. The Application User sees the modeling system as a black box with a defined interface to communicate. Since every application needs different properties of the modeled user, there is a need to adapt the modeling system to the application needs.

Administrator

The last user scenario is the Administrator. The Administrator is responsible for the system maintenance. Tasks are to start the system, to control the system during runtime and if improvements can be made, to carry them out. Improvements are in this scenario not improvements of code but improvements regarding the system. Since the modeling system is based on a service-oriented architecture (SOA) it is possible to distribute services over a local network. The Administrator should monitor the communication and computing loads of the affected computers and find an appropriate system structure.

The Administrator is a combination of software system and persons. The Administrator software system is for example the runtime environment which is responsible for starting the needed services and keep them running. Persons are responsible for more sophisticated tasks like managing the system's structure and monitoring system loads. Administrators are directly affected by the handling criteria and, as

already stated above, in charge of clustering and distribution of the modeling system.

The definition and description of user scenarios help to compare systems under different points of view. Aspects and criteria which are examined during the evaluation are described in the following section.

8.6.2 Evaluation Criteria

In addition to user scenarios, criteria are used to evaluate the two approaches. A criterion describes a standard or an aspect against which something is measured. After the measurement, the evaluation result of the different approaches is compared. An objective comparison can be provided by assigning values to each criteria according to the measurement result, but many criteria allow only a qualitative description of the criteria satisfaction.

The first step is to find out the relevant evaluation criteria. The following criteria are listed and described in [Shehory and Sturm 2001; Chung *et al.* 2000; Hetzel 1988]. But not all criteria listed in literature are applicable and useful in this case. Basically, the proposed and implemented modeling system of this thesis project can be evaluated in a similar way to distributed systems, since Openwings is based on the network layer and allows to run communicating services on different platforms without further restrictions. Further, relevant aspects in the field of user modeling must be added to the technical evaluation criteria for distributed systems (e.g. privacy, security, adaptability, etc.). The following section lists and describes the used criteria:

- Performance
- Reliability
- Clustering and Distribution
- Handling
- Adaptability
- Scalability
- Privacy and Security

Performance

Not all software systems have explicitly specifications on performance issues. But every system will have implicit performance requirements. The software should not take infinite time or need infinite resources to execute. Performance evaluation of a software system usually includes: resource usage, throughput, stimulus-response time and queue lengths detailing the average or maximum number of tasks waiting to

be serviced by selected resources. The goal of performance testing can be for example, performance bottleneck identification, performance comparison and evaluation. [Hetzel 1988]

Poor performance is often the result of problems in the architecture or design rather than the implementation. This statement is reinforce by the following quote:

> *"Performance is largely a function of the frequency and nature of inter-component communication, in addition to the performance characteristics of the components themselves, and hence can be predicted by studying the architecture of a system."* [Clements 1996]

Therefore, the evaluation of the system performance is made by examining the architecture and the communication processes of the solution approaches.

Reliability

The reliability of a software component is the degree to which it can function correctly in the presence of exceptional inputs or stressful environmental conditions. [IEEE 1991] defines reliability as

> *"... the ability of a system or component to perform its required functions under stated conditions for a specified period of time."*

There are no concerns about the functional correctness of the software addressed by the reliability criterion. Only problems such as machine crashes, process hangs or abnormal termination are monitored.

Clustering and Distribution

Clustering is a method to arrange multiple computers in form of a cluster. This group of computers or servers acts like a single system. [Whatis.com 2005]

The distribution of software as a result of clustering is an important issue, because software component should not be divided arbitrarily. Thus, aspects like load balancing are used to find the best distribution.

Handling

Handling describes the manageability of the software system. Manageability covers the activity of ensuring that the application is alive and functioning normally, as well as that of checking that an application performance is as expected. [Whatis.com 2005]

Concerning the different user scenarios, there are different handling aims. For example, the Application User wants to access the system through a simple interface, or the Administrator wants to manage a flexible and small system.

Adaptability

Adaptations describe changes of the system in order to accommodate changes of the environment. Adaptation of software systems are caused by changes from an old environment to a new one. If adaptability is provided by the software system this change results in a new system, which should meet the needs of the new environment. [Chung and Subramanian 2001]

Scalability

Scalability is the ability of a software application to change its size. In most cases the rescaling is done to a larger size. [Whatis.com 2005]

Privacy and Security

Software quality, reliability and security are related to each other. Flaws in software or software with a bad quality can be exploited by intruders for unauthorized access to the system. The purpose of security testing includes identifying and removing this software flaws. To find vulnerabilities, simulated security attacks are performed. [Hetzel 1988]

The description of these evaluation criteria helps to examine the approaches. For both systems the same definitions must be applied in order to ensure a comparable result. To allow the assessment of the systems concerning the evaluation criteria, it is necessary to use a good evaluation setup. The used evaluation setup is described in the following section.

8.6.3 Evaluation Setup

A evaluation setup describes the configuration and conditions under which the evaluated systems are operating. The major point for designing an evaluation setup is that it must include the needed features to allow a conclusion about all relevant evaluation criteria. For example, if the performance in form of access time is measured, a setup is needed, where the access to a particular system is performed several times.

Since this is not a holistic evaluation, but rather a comparison of the two solution approaches, the evaluation is limited on mainly qualitative descriptions of the criteria. Thus, also the evaluation setup represents a simple approach to an evaluation setup for an extensive evaluation.

The evaluation setup used for this thesis consists of two external systems and the GUI visualization service. These three services communicate with the modeling system at the same time and are working in a concurrent manner. This allows to assess the performance of the modeling systems. Figure 8.13 shows the class diagram of one external system (PerformanceTest1). All services are running on the same "Openwings 1.1" platform.

The classes `PerformanceTest1` and `PerformanceTes2` are wrapped into Openwings services. After starting the service, they communicate with the modeling

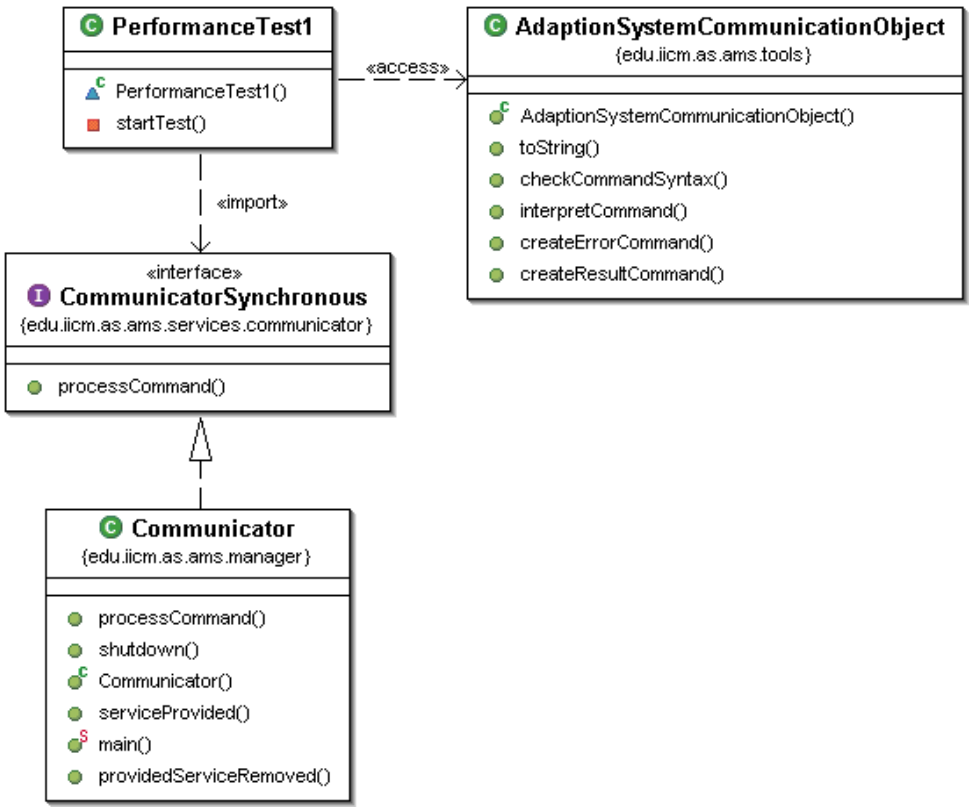

Figure 8.13: UML class diagram of the evaluation setup (performance test)

system by connecting to the `Communicator`, respectively to the `Manager` service. The service `PerformanceTest1` writes a user parameter (last name) to the user profile while the second `PerformanceTest2` service queries the WAVI model of the same user. Both commands are repeated 50 times and the mean processing time is written to the log windows of the services.

As described in Section 7.5, Openwings allows to secure the transport and the services from unauthorized access. During the deployment of a service it is possible to set the security. Either security is enabled or disabled without any change to the Java source. Since security is an important issue for user modeling systems the consequences of enabling Openwings security to the performance criterion is examined by repeating the same evaluation process with enabled secure communication.

Applying the evaluation setup of this section returns the evaluation results. These results are depicted in the following section.

8.6.4 Evaluation Results

The evaluation results are structured regarding user scenarios, since the three user scenarios give different results for the evaluation criteria. In many cases, evaluation criteria are not relevant for a specific user scenario. For example, the Developer is not affected by the privacy criterion. The Developer is responsible to fulfill the

privacy requirements, but the evaluation of the privacy criterion is not relevant for him.

The first user scenario is the Developer, followed by the Application User. Finally, the Administrator scenario is examined in the following section.

Developer

As already stated in Sub-section 8.6.1, the task of the Developer is to fulfill the user requirements and create the needed system. During the design and the implementation process the Developer is interested in two evaluation criteria, namely the *Handling* and the *Scalability* of the approach.

The *Handling* of the solution approaches is different for the Developer. Since the micro-approach contains 15 services, a well organized development environment is needed to cope with the source code during the implementation. Compared to the macro-approach, where only 4 services are used, this is a drawback of the micro-approach regarding the handling.

Comparing *Scalability* of the two solution approaches does not allow to prefer one approach. Scalability of the macro-approach is implemented by adding additional classes to the related service and changing the service contracts. Performing a scalability change for the micro-approach is handled completely different. Not additional classes are added to existing macro-services but new classes are wrapped by new services. To allow the access to this new services the common services must know about them and therefore modified on the source level. Therefore, from the viewpoint of the Developer scenario the scalability needs less efforts for the macro-approach.

Application User

The Application User is the user scenario for which the approaches are designed. Application Users are interested in aspects like Performance, Robustness, Adaptability and Privacy/Security.

Regarding *Performance*, comparing the architecture of the two solution approaches makes clear, that the micro-approach needs more inter-service communication to operate. Since this communication is processed by applying the Java RMI technology (see Sub-section 7.4.4), there is a lot of communication overhead. This reduces the system performance of the micro-approach. Considering the macro-approach, only 4 services are communicating with each other, which consequently allows to to prefer the macro-approach.

The result of this performance inspection by using the evaluation setup (see Sub-section 8.6.3) results in command processing times, which are shown in Table 8.1.

The mean command processing time for unsecure communication is for the micro-approach about four times higher than for the macro-approach. This is arguable with the more inter-service communication of the micro-approach. There is not much difference between the two commands (Com1 and Com2) although different parts of the system are affected. Further, it can be noticed, that the data caching of the WAVI modeler increases the processing speed for this command. The first execution

	Micro-approach		Macro-approach	
Connectors	Com1	Com2	Com1	Com2
unsecure	96 [ms]	104 [ms]	18 [ms]	21 [ms]
secure	742 [ms]	756 [ms]	155 [ms]	163 [ms]

Com1: Set User Information
Com2: Get WAVI Model

Table 8.1: Mean Command Processing time for Micro- and Macro-approach in milliseconds

of Com2 was in all cases the slowest, since in this case, the modeler has to connect to the profiler in order to initialize the WAVI model. All further requests for the WAVI model are answered by using the cached data within the Modeler. Securing the communication between the services increases the processing time enormously and hinders a reasonable utilization of the micro-approach. The processing time is nearly one second for each command by the micro-approach and more than 150 millisecond for the macro-approach.

Over all, the macro-approach with its less inter-service communications is the preferable solution approach regarding performance.

Considering *Reliability*, the life cycle of services running on the Openwings framework is monitored by the framework. Openwings detects if a service has stopped and tries to restart the service again. Nevertheless, error handling during runtime has to be implemented by the approaches itself. A service-based architecture requires an error handling within services. Therefore, each service must be able to deal with occurring errors. Since both approaches implement the same functionality also the error handling mechanism is the same. Thus, there is no difference regarding Reliability between micro- and macro-approach although Openwings has to handle 15 services for the micro-approach and only 4 services for the macro-approach.

Regarding *Adaptability*, the modeling system can be adapted in form of adding profilers and user models. Each Application User is able to define its own profilers and modelers. Both approaches are designed to allow such adaptation and therefore they are considered as coequal regarding Adaptability.

Both approaches apply the same *Privacy and Security* techniques. Privacy is provided by using pseudonyms for users and by the privacy modul, which allows to define privacy levels for parts of the user information.

Security is established by utilizing the Openwings security features. Thus, there is no difference between the two approaches concerning privacy and security.

Administrator

According to the definition of the Administrator scenario (see Sub-section 8.6.1), criteria like Robustness and Stability, Clustering and Distribution, Handling and Scalability are in the interest of an Administrator.

Regarding *Reliability*, the Administrator is responsible to deal with outtakes of the system. Outtakes are for example the crash of a service. In this case the

Administrator has to identify the crashed service and restart it. This task is mainly performed by the runtime environment. Since the services itself are designed to be stateless, there is no rollback problem. In the case of a crashed service, the Application User receives an error message and is asked to resubmit the command. The runtime environment has to monitor a different amount of services, concerning the two approaches. The micro-approach needs 15 services and this makes it more error prone compared to the macro-approach, where only 4 services are needed. This fragility of the micro-approach results in more work for the Administrator.

A further task of the Administrator is to keep track of *Clustering and Distribution* in order to provide load balancing. Clustering techniques provided by the Openwings framework are applied to spread the processing load over several machines. Since the micro-approach offers more possibilities to distribute the complete system, it allows to balance the load in a more accurate way. However, the macro-approach also allows to distribute the system but not in such a precise manner as the micro-approach.

Considering outtakes and distribution issues, the *Handling* of parts of the systems is important. The micro-approach with its higher amount of services needs more resources to administer them, and therefore the Handling is not that easy as for a system with less services.

To provide the required performance of the modeling system, the *Scalability* is an important aspect. The Administrator has the possibility with clustering and distribution methods to adjust the Scalability of the system. Comparing the Scalability of the micro-approach with that of the macro-approach shows advantages for the first one. For example, it is possible to place one small component of the system (e.g. `UserInformationHandler`) on another platform if it is under heavy load. With the macro-approach this is not possible because there is only one service representing the `Profiler`.

Evaluation Summary

In order to summarize the evaluation results, this last part of the chapter condenses the main outcomes in form of two tables (see Table 8.2 and Table 8.3).

Each table describes the evaluation results for one solution approach. Further, by assigning numerical values to strengths and weaknesses the author of this thesis has tried to compare both approaches in a quantitative way.

Micro-approach			
	Developer	Application User	Administrator
Performance		-2	
Reliability		-1	-1
Clustering			2
Handling	-1		-2
Adaptability		0	
Scalability	0		1
Privacy/Security		0	

-2 ... strong drawback; -1 ... weak drawback; 0 ... equally
1 ... weak advantage; 2 ... strong advantage; _ ... not affected

Table 8.2: Evaluation Summary for the Micro-approach

Macro-approach			
	Developer	Application User	Administrator
Performance		1	
Reliability		0	1
Clustering			0
Handling	0		1
Adaptability		0	
Scalability	0		0
Privacy/Security		0	

-2 ... strong drawback; -1 ... weak drawback; 0 ... equally
1 ... weak advantage; 2 ... strong advantage; _ ... not affected

Table 8.3: Evaluation Summary for the Macro-approach

Although the focus of the external systems (Application User) is different, it is possible to state that the macro-approach is preferable. Also for the user scenario Administrator, the macro-approach has advantages compared to the micro-approach. Only for Developers is not a big difference between the two solution approaches.

This section described the evaluation and comparison of micro- and macro-approach by depicting the evaluation process in form of user scenarios, evaluation criteria and an evaluation setup. The end of this section depicted the outcomes of the evaluation and summarized the main results.

The following section gives a summary of this chapter including comments on the design process and evaluation results.

8.7 Summary

The design process is an essential stage in every software development process. It that sense, it has to be emphasized that a good software design allows to satisfy the current needs of the application user and allows to extend the system for future changes. The demands to the modeling system are described in form of requirements in Section 8.2 followed by the architectural design and the specification of use cases in Section 8.3.

The functional design and the implementation is separated into the macro-approach and the micro-approach (see Section 8.4 and 8.5). In these sections the functional architecture of the approaches is introduced followed by the description of the implementations. The actual implementation does not cover the complete design outcome, and represents therefore a “first prototype”.

In the last section of this chapter the evaluation and comparison of the two approaches (see Section 8.6) were described. The evaluation results show that the macro-approach has some advantages compared to the micro-approach. Less inter-service communication efforts and a simpler system architecture provides more performance and better handling.

The following chapter gives a conclusion of this thesis and depicts some interesting aspects for further work in form of an outlook.

9. Summary and Outlook

Information about the user, in form of an internal representation of user traits and states, is an important matter for adaptive systems. Without any knowledge, or with wrong knowledge about the user, it is not possible for user-adaptive systems to perform accurate adaptations. Therefore, user information must be available to adaptive systems. The unit where this information is managed is called user model and represents all aspects of the system's idea about the user.

This thesis aims at a proposal for a modeling system by describing and examining important characteristics and requirements for such a system. The main concern of the author of this work was to create a versatile and secure user modeling system. Prerequisites for a versatile system are a well-organized and well-structured user profile and user model. Security for user modeling systems is often related to privacy since user information stored in a system is included by the right to privacy.

The proposed system is based on a service-oriented architecture and therefore, consequences of a service-oriented architecture regarding user modeling systems and especially on the main concerns of this work were examined. In general, the following aspects were considered and investigated within this thesis:

- terminology,
- modeling techniques,
- user modeling standards,
- available user modeling systems,
- privacy and security techniques, and
- service-oriented architecture.

Due to these aspects, this thesis is divided into a theoretical part and a practical part. The theoretical part examines the listed aspects, while the practical part uses the outcome of the examination and describes the realization of these aspects in form of a service-based solution approach.

In order to be able to describe important issues regarding user modeling it is previously necessary to clarify the *terminology*. In literature the meaning of the terms user modeling, user profiling and adaptive e-learning systems differs. Within this thesis, the term user profiling is used in relation to a low-level user model where basically user information is collected in form of "raw data". The user model exploits the user profile and infers semantically richer user information. The utilization of user profiles and user models within adaptive e-learning systems is described by examining available theoretical approaches and possible types of systems.

Distinct adaptive e-learning systems apply different user models, but all of them use a kind of learner or student model. Considering a learner model, aspects like the demands to a learner model, available *user modeling techniques* and the content of a learner model were described. In order to offer information about a learner, a modeling system must be able to construct, initialize and update the learner model. For

each of these steps several methods and techniques can be identified (e.g. Bayesian Methods, machine learning methods, overlay and stereotype methods, etc.). The content of learner models can be separated into domain specific and domain independent information. Domain specific information describes the learner's expertise regarding a knowledge domain, while domain independent information covers learner traits, such as preferences, goals and so forth.

The applicability of already existing solutions to describe the content of a learner model was examined in form of an inspection of available *user modeling standards*. There exist mainly three standards which are suitable, namely GESTALT, PAPI Learner and IMS LIP. The standards PAPI Learner and IMS LIP are seen as the most valuable and allow to model the user in an appropriate level of detail. Although these standards are not utilized in many systems, this thesis gives a clear proposal for utilizing them since they allow to organize the content of the user model in a well-defined way and offer precautions for privacy and security issues.

The second examination of existing solutions was the description of *available user modeling systems*. From a chronological viewpoint three types of user modeling systems were examined, namely Early User Modeling System, Shell Systems and User Modeling Servers. Latest developments apply Web Service and agent technologies, which enable flexibility in form of the ability to add further modeling components.

Prior to the practical part of this work, some aspects concerning the design and the implementation of the proposed modeling system were examined. Reflections for the proposed solution were concerning *privacy and security techniques*, organization and structure of user profiles and user models as well as *service-oriented architecture*. In order to provide an exploitable user profile, it is necessary to organize the user information contained in the profile. This is possible by splitting the user profile into several sub-profiles where each sub-profile handles a specific type of user information. The user model is organized in a similar way. This organization facilitates the utilization of privacy and security mechanisms since each sub-profile and sub-model can be connected with a particular privacy level (e.g. demographic data is only readable). Security issues must be enabled by the applied service-oriented framework. Two frameworks were examined and compared, namely OSGi and Openwings. *Openwings* was chosen to be used for the design and implementation of the modeling system and represents a specification of a service-oriented framework.

The core of the practical part of this theses is the design and implementation of the *solution approach*. The proposed solution represents a user modeling system based on the Openwings framework and integrates the deductions of all previously described aspects. Concerning software development patterns for service-oriented design, there are no specific rules about how much functionality should be offered by one service. The practical part of this thesis worked on this question in form of comparing two extrema, namely a macro-approach and a micro-approach. The macro-approach groups all related functional components and wraps them into one service. This approach results in a smaller amount of services compared to the micro-approach, where each component is wrapped by one service.

Using a service-oriented architecture as the basis for a user modeling system allows to realize modular and therefore versatile user profiles and user models. There-

fore, the implemented service-oriented system utilizes an up-to-date software design paradigm, which is often referred to as the successor of the object-oriented paradigm. Privacy concerns are addressed by applying methods like pseudonymity combined with a security service, which is responsible for authentication and access restrictions. Communication security and the prevention of intrusion is handled by the Openwings framework. That emphasizes the advantages of this service-oriented framework. Comparing the micro-approach and the macro-approach shows different results. There is no clear answer to the investigated design question. The appropriate design must be chosen depending on the most relevant required criteria, whereby this comparison is helpful and may be used as a reference.

Extensions and improvements of the proposed solution are foreseen through the research project AdeLE. Especially privacy and security concerns combined with the enhancement of the scrutability of the user model might be considered for further work. Scrutable user models allow the user to view and change his user model, and therefore, gives the user more freedom and control concerning the adaptation and presentation of the learning content. Further reasons for a scrutable user model is the traceability of adaptations by showing the users reasons for the current adaptation of the learning material and offering "the other alternatives" (those not selected by the system).

Additional further work may aim to integrate available user modeling standards into the modeling system. This integration may take place in form of an additional modeler, which uses the data from the user profile and creates the specified presentation according to the specification of the standard.

A. Additional UML Diagrams

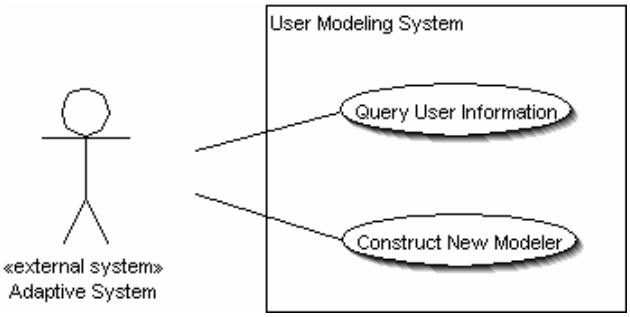

Figure A.1: Use Cases for the Modeler

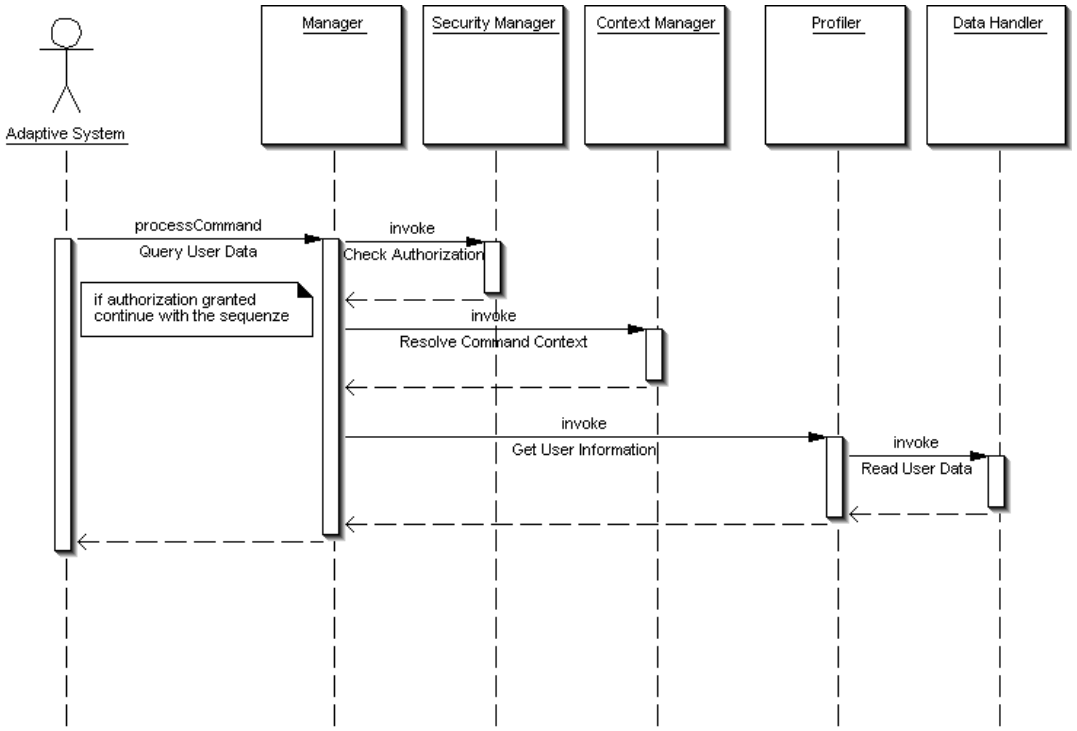

Figure A.2: Sequence Diagram for querying the User Profile

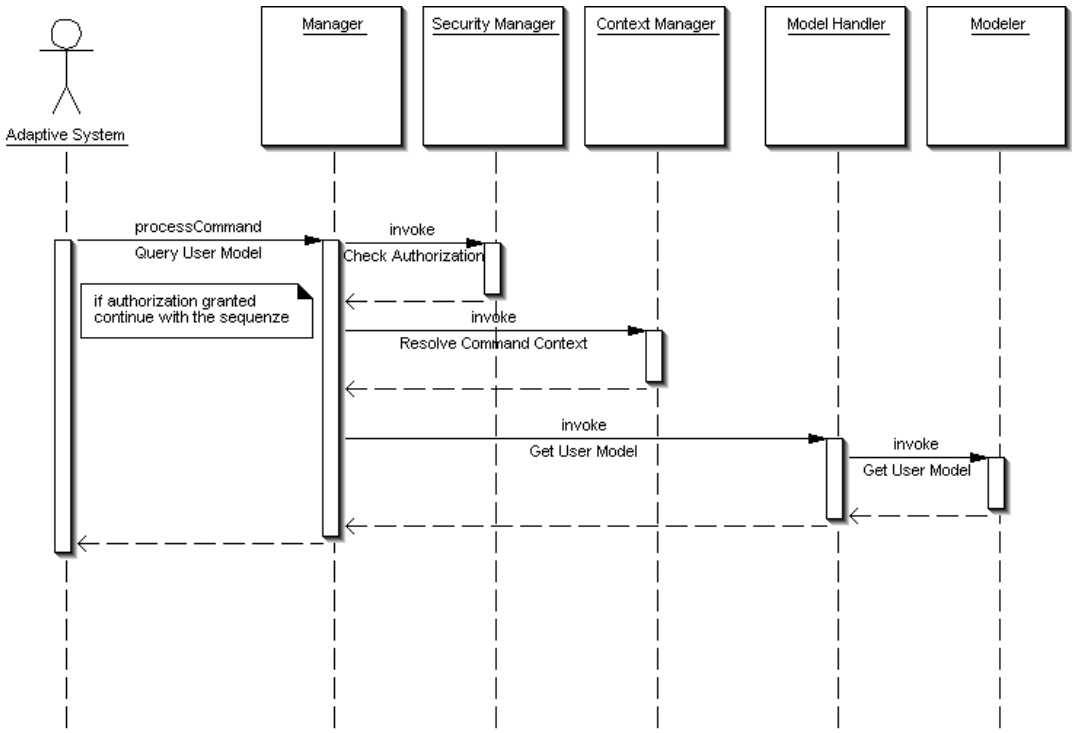

Figure A.3: Sequence Diagram for querying the User Model

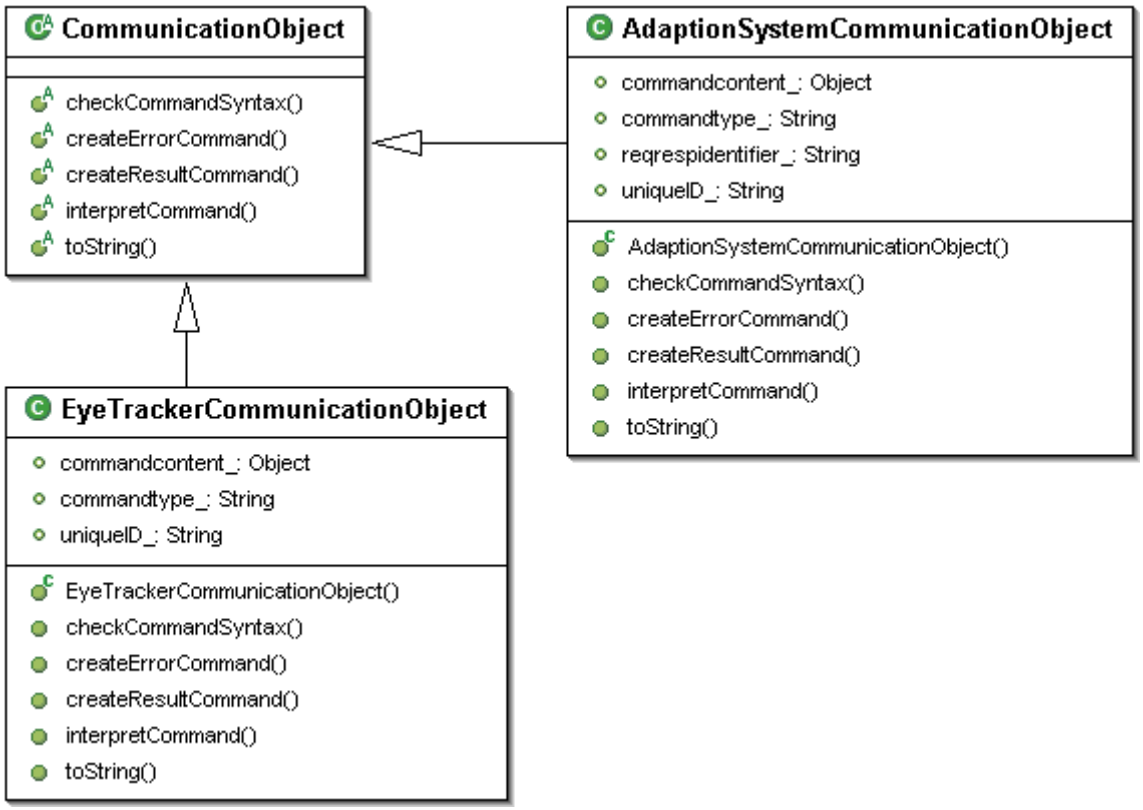

Figure A.4: UML class diagram of the abstract class *CommunicationObject* with two available implementations, namely *EyeTrackerCommunicationObject* and *AdaptionSystemCommunicationObject*

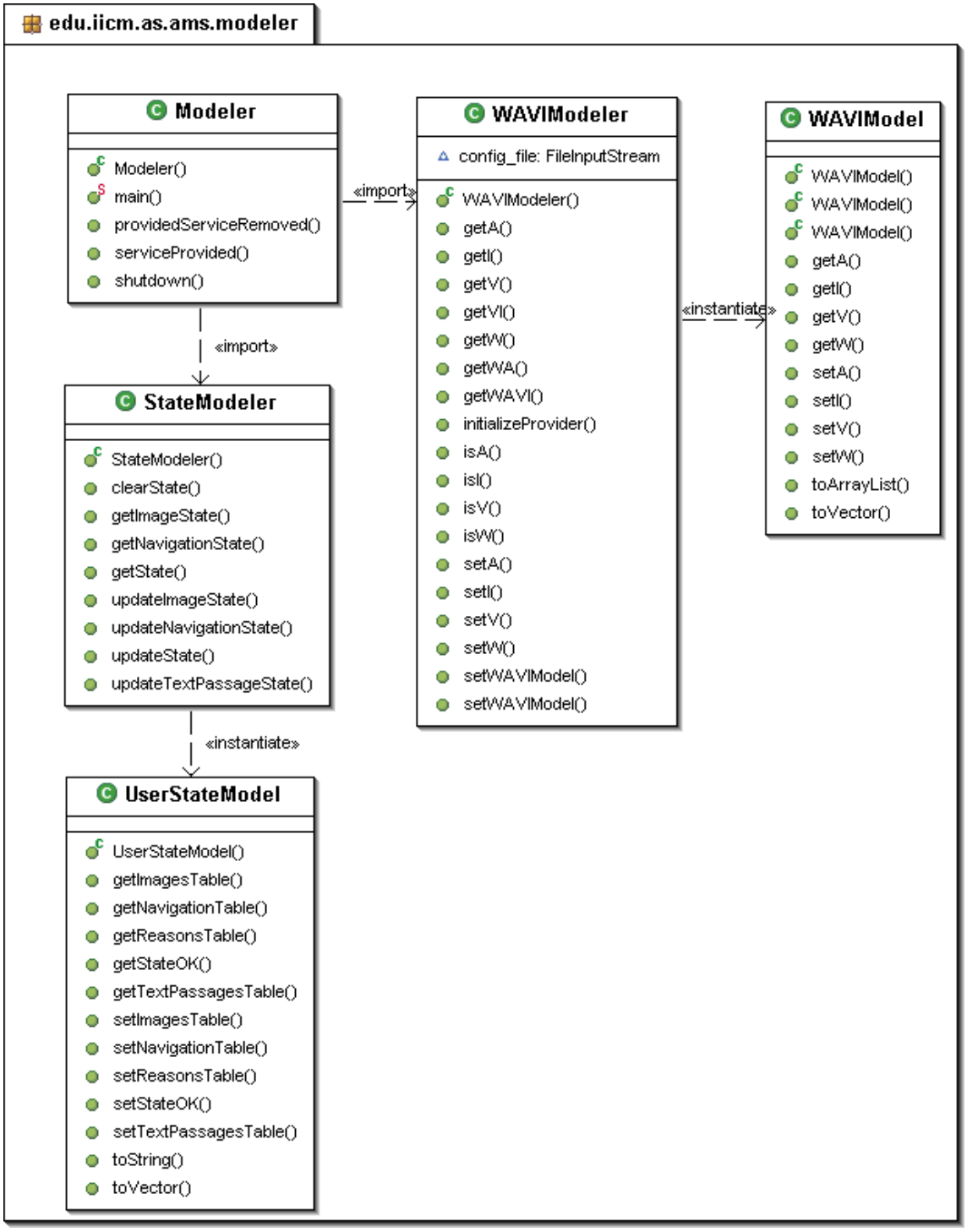

Figure A.5: UML Class Diagram of the *Modeler* Component (macro-approach)

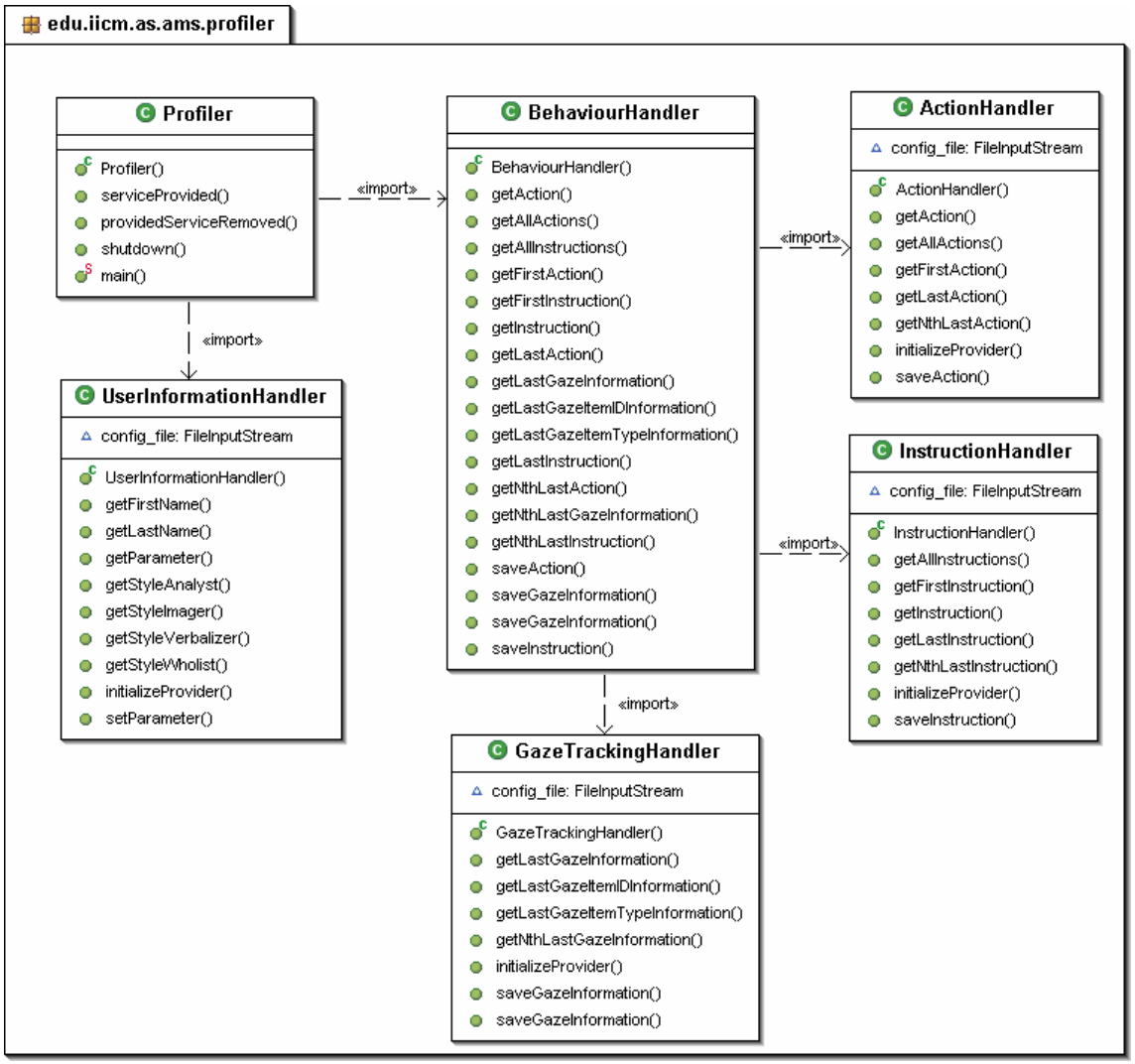

Figure A.6: UML Class Diagram of the *Profiler* Component (macro-approach)

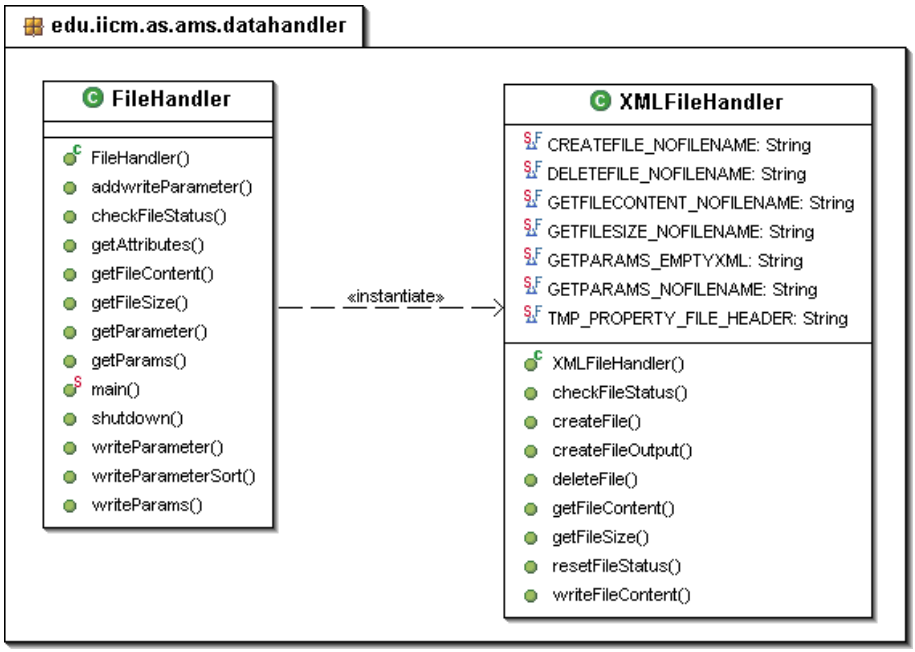

Figure A.7: UML Class Diagram of the *DataHandler* Component (macro-approach)

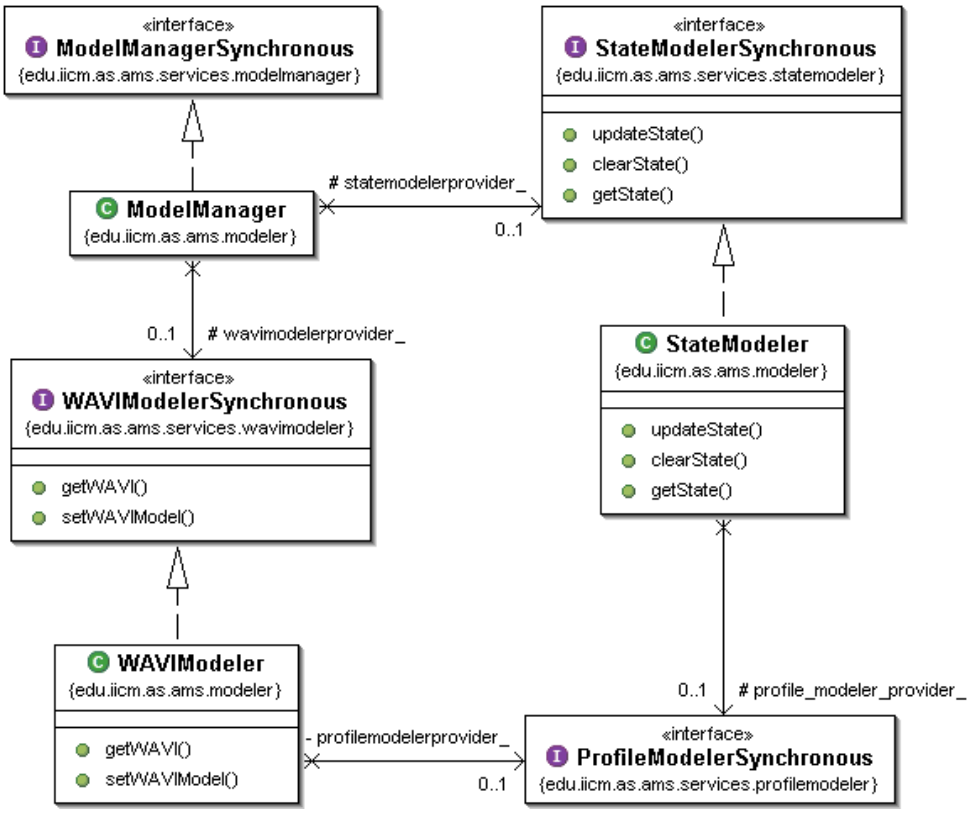

Figure A.8: UML class diagram including services which are connected to the *ModelManager* service (micro-approach)

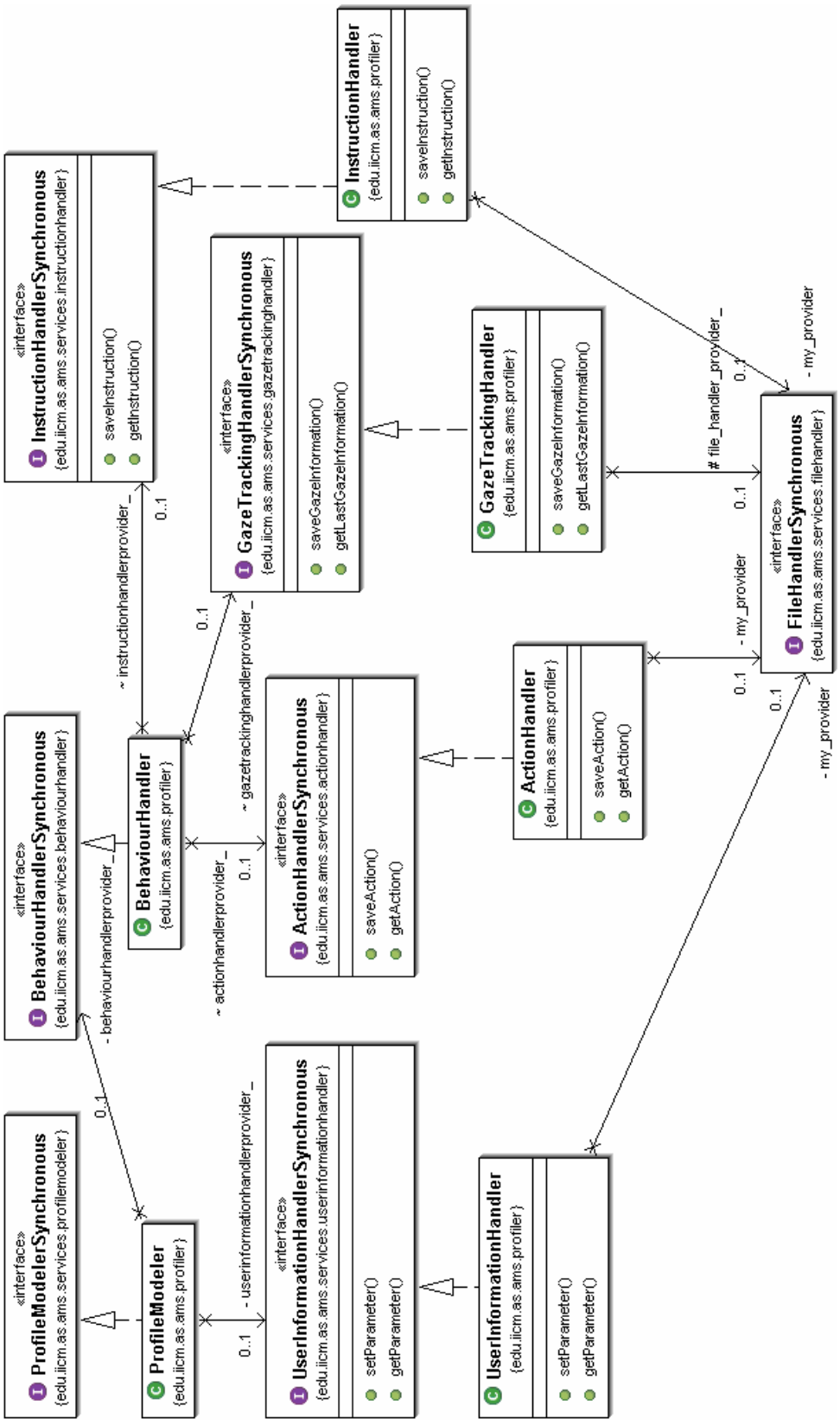

Figure A.9: UML class diagram including services which are connected to the *ProfileModeler* service (micro-approach)

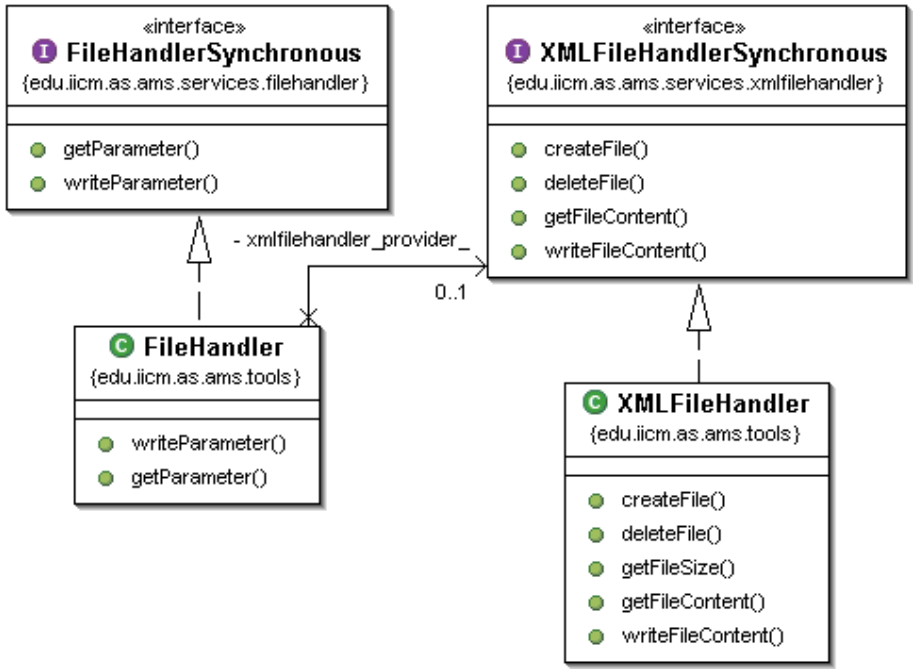

Figure A.10: UML class diagram including services which are connected to the *FileHandler* service (micro-approach)

B. Screen shots

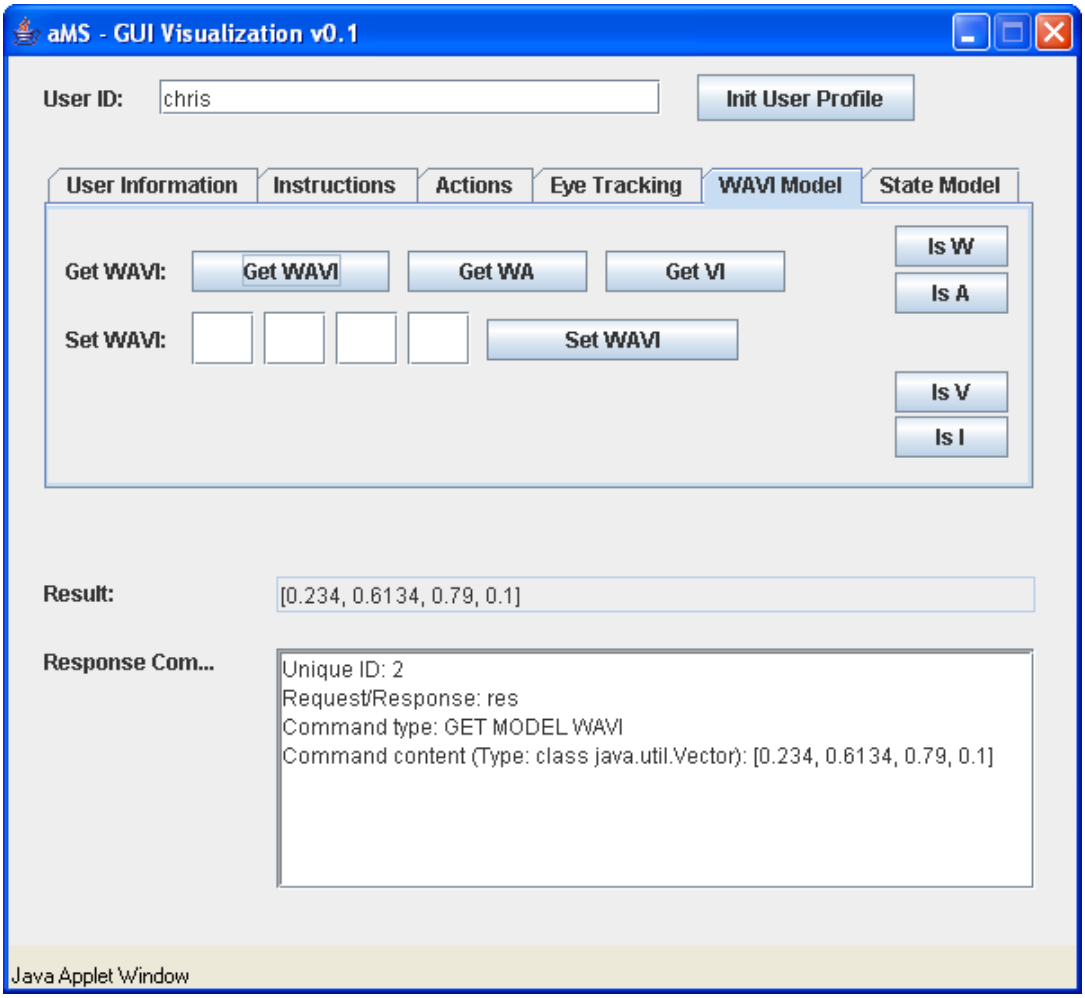

Figure B.1: Screen shot of the GUI Visualization, where the WAVI model of user "chris" is queried.

Figure B.2: Screen shot of the AdeLE front-end showing a course visualization.

List of Figures

1.1 Architecture of the AdeLE system [Gütl and Garcia-Barrios 2005] . . 5

2.1 User and User Model [Kay 2000b] 10
2.2 Adaptive System [Brusilovsky and Maybury 2002] 13
2.3 Components of an ITS [Brusilovsky 1994] 17

3.1 Role of the User Model in Adaptation [Kay 2000b] 21
3.2 Components of an ITS [Brusilovsky 1994] 23
3.3 Adaptive Hypermedia Techologies [Brusilovsky 2001] 26

4.1 GESTALT Functional Architecture [Wade *et al.* 2002] 44
4.2 Relationships among PAPI Learner parts [Farance 2001] 47
4.3 PAPI Learner Information Groups. [Farance 2001] 48
4.4 IMS LIP Core Segments [Smythe *et al.* 2001] 52
4.5 <learnerinformation> elements [Smythe *et al.* 2001] 54
4.6 <identification> elements [Smythe *et al.* 2001] 55
4.7 <transcript> elements [Smythe *et al.* 2001] 56
4.8 <securitykey> elements [Smythe *et al.* 2001] 57

5.1 GUMS Architecture [Finin and Drager 1986] 61
5.2 Internal view of the BGP-MS [Blank 1996] 64
5.3 Communication with BGP-MS [Kobsa and Pohl 1995] 65
5.4 Personis Collaboration Architecture [Kay *et al.* 2002] 69
5.5 Personis Internal Architecture [Kay *et al.* 2002] 70
5.6 Overview of the LDAP Server Architecture [Fink 2003] 71
5.7 Overview of the Agent-based Architecture [González *et al.* 2005] . . . 72

6.1 Application implementation layers: Services, components, objects [Endrei *et al.* 2004] 78
6.2 Directory service [Hashimi 2003] 78
6.3 Web Services [Kreger 2001] 80
6.4 OSGi framework [OSG 2003] 82
6.5 Openwings Architecture [Bieber and Carpenter 2001] 84
6.6 Structure of the Learner Profile 89

7.1 Architecture of the Openwings Framework [Bieber and Carpenter 2001] 93
7.2 Connector Architecture [Bieber and Carpenter 2001] 95
7.3 Component Services Use Cases [Bieber and Carpenter 2001] 96
7.4 Component Install State Diagram [Bieber and Carpenter 2001] 101
7.5 Openwings Policy Concept 106

8.1 Layer Architecture 118
8.2 Use Cases for the Profiler 119
8.3 Use Cases for the Eye-tracker 120

8.4 Functional Architecture of the Macro-approach 121
8.5 Sequence Diagram for *Send User Data* 124
8.6 Class Diagram of the Manager Component 125
8.7 Functional Architecture of the Manager 127
8.8 Functional Architecture of the Modeler 127
8.9 Functional Architecture of the Profiler 128
8.10 Functional Architecture of the Tools Component 129
8.11 Functional Architecture of the Micro-Approach (adapted from [Gütl and Garcia-Barrios 2005]) 130
8.12 *Communicator* class diagram of the micro-approach 131
8.13 UML class diagram of the evaluation setup 137

A.1 Use Cases for the Modeler 146
A.2 Sequence Diagram for querying the User Profile 146
A.3 Sequence Diagram for querying the User Model 147
A.4 UML class diagram of the *CommunicationObject* 148
A.5 UML Class Diagram of the *Modeler* Component 149
A.6 UML Class Diagram of the *Profiler* Component (macro-approach) . . 150
A.7 UML Class Diagram of the *DataHandler* Component (macro-approach) 151
A.8 *ModelManager* class diagram of the micro-approach 151
A.9 *ProfileModeler* class diagram of the micro-approach 152
A.10 *FileHandler* class diagram of the micro-approach 153

B.1 Screen shot of the the GUI Visualization 154
B.2 Screen shot of the AdeLE front-end 155

List of Tables

8.1 Mean Command Processing time for Micro- and Macro-approach in milli-seconds 139
8.2 Evaluation Summary for the Micro-approach 141
8.3 Evaluation Summary for the Macro-approach 141

Listings

4.1 IMS LIP example 53
4.2 IMS LIP <name> element 56
5.1 Primary sub-model in Doppelgänger [Orwant 1995] 67
5.2 Data from a sensor in Doppelgänger [Orwant 1995] 67
7.1 Creating Service Objects 103
7.2 Use and Discard Services 105
7.3 Subscribe and Unsubscribe to Services 105
8.1 XML user information file 132

Bibliography

[ACTS 2005] The Advanced Communications Technology and Services (ACTS) Program, 2005. http://www.cordis.lu/infowin/acts/analysys/intro/index.html [Last access February 21th, 2005].

[ATG 2005] Art Technology Group (ATG) Products, 2005. http://www.atg.com/products [Last access July 18th, 2005].

[Bieber and Carpenter 2001] Guy Bieber and Jeff Carpenter. Openwings, A Service-Oriented Component Architecture for Self-Forming, Self-Healing, Network-Centric Systems (Rev 2.0), 2001. http://www.openwings.org/download/specs/openwingswp.pdf [Last access March 18th, 2005].

[Bieber and Carpenter 2003] Guy Bieber and Jeff Carpenter. Openwings Connector Service Specification Ver 1.0 Final, 2003. http://www.openwings.org/download/specs/Openwings_Connector_Services.pdf [Last access March 18th, 2005].

[Bieber and Crumpton 2003] Guy Bieber and Kathleen Crumpton. Openwings Install Service Specifcation Ver. 1.0 Final, 2003. http://www.openwings.org/download/specs/Openwings_Install.pdf [Last access April 6th, 2005].

[Bieber and Thrash 2003] Guy Bieber and Brian Thrash. Openwings Security Specification Ver. 1.0 Final, 2003. http://www.openwings.org/download/specs/Openwings_Security.pdf [Last access March 18th, 2005].

[Bieber *et al.* 2003] Guy Bieber, Mark Nelson, and Lon Chang. Openwings Interface Definition Specification Ver. 1.0 Final, 2003. http://www.openwings.org/download/specs/openwings_interface.pdf [Last access March 18th, 2005].

[Blank 1996] Karlheinz Blank. Benutzermodellierung für adaptive interaktive Systeme: Architektur, Methoden, Werkzeuge und Anwendungen. PhD thesis, University of Stuttgart/Germany, 1996.

[Bødker 2000] Susanne Bødker. Scenarios in User-Centered Design - Setting the Stage for Reflection and Action. Interacting with Computers, vol. 13, p.p. 61–75, 2000. http://www.clab.edc.uoc.gr/application/scenarios_in_user-center.pdf [Last access September 18th, 2005].

[Booth *et al.* 2004] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture. Technical report, World Wide Web Consortium (W3C), 2004. http://www.w3.org/TR/ws-arch/ [Last access September 18th, 2005].

[Brajnik and Tasso 1994] Giorgio Brajnik and Carlo Tasso. A shell for developing non-monotonic user modeling systems. International Journal of Human-Computer Studies, vol. 40, no. 1 p.p. 31–62, 1994.

[Brusilovsky and Maybury 2002] Peter Brusilovsky and Mark T. Maybury. From adaptive hypermedia to the adaptive web. Communications of the ACM, vol. 45, no. 5 p.p. 30–33, 2002.

[Brusilovsky 1994] Peter Brusilovsky. The Construction and Application of Student Models in Intelligent Tutoring Systems. Journal of Computer and System Sciences International, vol. 32, no. 1 p.p. 70–89, 1994. `http://www2.sis.pitt.edu/~peterb/papers/studentmodels.pdf` [Last access May 18th, 2005].

[Brusilovsky 1996] Peter Brusilovsky. Methods and Techniques of Adaptive Hypermedia. User Modeling and User-Adapted Interaction, vol. 6, no. 2–3 p.p. 87–129, 1996. `http://www2.sis.pitt.edu/~peterb/papers/UMUAI96.pdf` [Last access May 18th, 2005].

[Brusilovsky 1998] Peter Brusilovsky. Adaptive Educational Systems on the World-Wide-Web: A Review of Available Technologies. In Proceedings of workshop WWW-Based Tutoring at 4th International Conference on Intelligent Tutoring Systems (ITS'98), 1998.

[Brusilovsky 2001] Peter Brusilovsky. Adaptive Hypermedia. User Modeling and User-Adapted Interaction, vol. 11, no. 1–2 p.p. 87–110, 2001. `http://www2.sis.pitt.edu/~peterb/papers/brusilovsky-umuai-2001.pdf` [Last access May 18th, 2005].

[Caglayan *et al.* 1997] Alper Caglayan, Magnús Snorrason, Jennifer Jacoby, James Mazzu, Robin Jones, and Krishna Kumar. Learn Sesame - A Learning Agent Engine. Applied Artificial Intelligence, vol. 11, no. 5 p.p. 393–412, 1997. `http://www.aminda.com/mazzu/AAI97_LearnSesame.pdf` [Last access July 18th, 2005].

[Carpenter and Bieber 2003] Jeffrey Carpenter and Guy Bieber. Openwings Component Service Specification Ver 1.0 Final, 2003. `http://www.openwings.org/download/specs/Openwings_Component_Services.pdf` [Last access March 18th, 2005].

[Castillo *et al.* 2003] Gladys Castillo, Joao Gama, and Ana M. Breda. Adaptive Bayes for a Student Modeling Prediction Task based on Learning Styles. In Proceedings of the 9th International Conference on User Modeling (UM'03), P.p. 328–332, 2003. `http://www.mat.ua.pt/gladys/Papers/UM2003.pdf` [Last access June 18th, 2005].

[Chung and Subramanian 2001] Lawrence Chung and Nary Subramanian. Process-Oriented Metrics for Software Architecture Adaptabilit. In Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE'01), P.p. 310–311, Washington, DC, USA, 2001. IEEE Computer Society. `http://www.utdallas.edu/~chung/ftp/POMSAA-Poster.pdf` [Last access September 21th, 2005].

[Chung *et al.* 2000] Lawrence Chung, Brain A Nixon, Eric Yu, and John Mylopoulos. Non-Functional Requirements in Software Engineering. International Series in Software Engineering Volume 5. Kluwer Academic Publishers, 2000.

[Clements 1996] Paul C. Clements. Coming attractions in software architecture. Technical Report No. CMU/SEI-96-TR-008, Software Engineering Institute, Carnegie Mellon University, 1996. `http://www.sei.cmu.edu/pub/documents/96.reports/pdf/tr008.96.pdf` [Last access September 13th, 2005].

[Conlan *et al.* 2002a] Owen Conlan, Declan Dagger, and Vincent Wade. Towards a Standards-based Approach to e-Learning Personalization using Reusable Learning Objects. In Proceedgins of the World Conference on E-Learning in Corporate, Government, Healthcare and Higher Education (E-Learn 2002), P.p. 210–217, September 2002. `http://www.cs.tcd.ie/~oconlan/publications/eLearn2002_v1.24_Conlan.pdf` [Last access February 13th, 2005].

[Conlan *et al.* 2002b] Owen Conlan, Vincent P. Wade, Catherine Bruen, and Mark Gargan. Multi-model, Metadata Driven Approach to Adaptive Hypermedia Services for Personalized eLearning. In Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'02), P.p. 100–111, 2002. `https://www.cs.tcd.ie/Owen.Conlan/publications/AH2002v0.99e11_Conlan.pdf` [Last access September 13th, 2005].

[Corno and Snow 1986] L. Corno and R.E. Snow. Adapting teaching to individual differences among learners. Handbook of research on teaching, 1986.

[Cranor *et al.* 2002] Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, and Joseph Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. World Wide Web Consortium (W3C), 2002. `http://www.w3.org/TR/P3P/` [Last access September 13th, 2005].

[De Bra *et al.* 1999] Paul De Bra, Geert-Jan Houben, and Hongjing Wu. AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. In Proceedings of the 10th ACM Conference on Hypertext and Hypermedia (HT'99), P.p. 147–156, 1999. `http://www.win.tue.nl/~debra/ht99/ht99.ps` [Last access June 7th, 2005].

[De Bra 2000] Paul De Bra. Pros and Cons of Adaptive Hypermedia in Web-Based Education. Journal on CyberPsychology and Behavior, vol. 3, p.p. 71–77, 2000. A draft version of this article is available at `http://www.win.tue.nl/~debra/cyber.html` [Last access May 19th, 2005].

[Dietinger 2003] Thomas Dietinger. Aspects of E-learning Environments. PhD thesis, Graz University of Technology, 2003.

[Dolog and Nejdl 2003] Peter Dolog and Wolfgang Nejdl. Challenges and Benefits of the Semantic Web for User Modelling. In Proceedings of the AH2003 workshop at 12th World Wide Web Conference. User Modelling Conference 2003, June 2003.

http://www.learninglab.de/~dolog/pub/semanticwebandum.pdf [Last access February 13th, 2005].

[eduPerson 2004] Middleware Architecture Committee for Education, Directory Working Group (MACE-Dir), Internet2. Draft Revision of eduPerson Specification, 2004. http://www.educause.edu/eduperson/ [Last access February 13th, 2005].

[El-Khatib *et al.* 2003] Khalil El-Khatib, Larry Korba, Yuefei Xu, and George Yee. Privacy and Security in E-Learning. International Journal of Distance Education, vol. 1, no. 4 p.p. 1–19, 2003. http://iit-iti.nrc-cnrc.gc.ca/iit-publications-iti/docs/NRC-45786.pdf [Last access August 13th, 2005].

[Endrei *et al.* 2004] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Poel Krogdahl, Min Luo, and Tony Newling. Patterns: Service-Oriented Architecture and Web Services. Chapter 2 Service-oriented architecture, RedBooks, 2004. http://www.redbooks.ibm.com/redbooks/SG246303/wwhelp/wwhimpl/js/html/wwhelp.htm [Last access August 13th, 2005].

[Far and Hashimoto 2000] Behrouz Homayoun Far and A.H. Hashimoto. A Computational Model for Learner's Motivation States in Individualized Tutoring System. In Proceedings of the 8th International Conference on Computers in Education (ICCE 2000), P.p. 21–24, 2000.

[Farance 2001] Frank Farance. PAPI Learner, Draft 8 Specification, 2001. http://edutool.com/papi/ [Last access February 13th, 2005].

[Finin and Drager 1986] Tim Finin and David Drager. GUMS: a General User Modelling System. In Proceedings of the Canadian Society for Computational Studies of Intelligence 1986 (CSCSI-86), 1986. http://acl.ldc.upenn.edu/H/H86/H86-1021.pdf [Last access June 13th, 2005].

[Fink 2003] Josef Fink. User Modeling Servers - Requirements, Design, and Evaluation. PhD thesis, University of Duisburg/Germany, 2003. http://www.ics.uci.edu/~kobsa/phds/fink.pdf [Last access July 21th, 2005].

[Gong 2003] Li Gong. Java 2 Platform Security Architecture, Version 1.2, 2003. http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc.html [Last access February 21th, 2005].

[González *et al.* 2005] Gustavo González, Cecilio Angulo, Beatriz López, and Josep Lluís de la Rosa. Smart User Models: Modelling the Humans in Ambient Recommender Systems. In Proceedings of the Workshop on Decentralized, Agent Based and Social Approaches to User Modelling (DASUM 2005), P.p. 11–20, 2005.

[GSS 2000] Sun Microsystems Inc. Java Generic Security Services (GSS), 2000. http://java.sun.com/j2se/1.4.2/docs/guide/security/index.html [Last access March 30th, 2005].

[Gütl and Garcia-Barrios 2005] Christian Gütl and Victor Manuel Garcia-Barrios. Towards an Advanced Modeling System applying a Service-based Approach. In Proceedings of the 5th IEEE International Conference on Advanced Learning Technologies (ICALT'05), P.p. 860–862, 2005. `http://www2.iicm.edu/cguetl/papers/ModelingSystem_ICALT05/ModelingSystem_ICALT05.pdf` [Last access September 21th, 2005].

[Han 2001] Binglan Han. Student Modelling and Adaptivity in web based Learning Systems. Master's thesis, Massey University/New Zealand, 2001.

[Hapner *et al.* 2002] Mark Hapner, Rich Burridge, Rahul Sharma, Joseph Fialli, and Kim Haase. Java Message Service API Tutorial and Reference. Addison-Wesley Longman Publishing Co., Inc., 2002.

[Hashimi 2003] Sayed Hashimi. Service-Oriented Architecture Explained. Article on the Internet, 2003. `http://www.ondotnet.com/pub/a/dotnet/2003/08/18/soa_explained.html` [Last access August 21th, 2005].

[Henze and Nejdl 2003] Nicola Henze and Wolfgang Nejdl. Logically Characterizing Adaptive Educational Hypermedia Systems. In Proceedings of International Workshop on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'03), P.p. 15–29. AH2003, 2003. `http://wwwis.win.tue.nl/ah2003/proceedings/paper2.pdf` [Last access June 1st, 2005].

[Hetzel 1988] Bill Hetzel. The Complete Guide to Software Testing, 2nd Edition. QED Information Sciences, Inc., 1988.

[IEEE 1991] IEEE. IEEE Standard Glossary of Software Engineering Terminology, 1991. IEEE Standard 610.12-1990.

[IMC 1996] Internet Mail Consortium (IMC). vCard - The Electronic Business Card Version 2.1, 1996. `http://www.imc.org/pdi/pdiproddev.html` [Last access February 12th, 2005].

[Jeremić and Devedžić 2004] Zoran Jeremić and Vladan Devedžić. Design Pattern ITS: Student Model Implementation. In Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT'04), P.p. 864–865, 2004. `http://csdl2.computer.org/comp/proceedings/icalt/2004/2181/00/21810864.pdf` [Last access October 1st, 2005].

[Jini 1999] Sun Microsystems Inc. Jini, 1999. `http://www.jini.org` [Last access February 21th, 2005].

[JNI 1997] Sun Microsystems Inc. Java Native Interface Specification, 1997. `http://java.sun.com/j2se/1.4.2/docs/guide/jni/spec/jniTOC.html` [Last access February 21th, 2005].

[Kashihara *et al.* 2000] Akihiro Kashihara, Kinshuk, Reinhard Oppermann, Rossen Rashev, and Helmut Simm. A Cognitive Load Reduction Approach to Exploratory Learning and Its Application to an Interactive Simulation-Based Learning System. Journal of Educational Multimedia and Hypermedia, vol. 9, no. 3 p.p. 253–276, 2000. Abstract available at `http://www.aace.org/dl/index.cfm/fuseaction/ViewPaper/id/6284/toc/yes` [Last access June 7th, 2005].

[Kay *et al.* 2002] Judy Kay, Bob Kummerfeld, and Piers Lauder. Personis: A server for user modeling. In Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'2002), P.p. 201–212, 2002.

[Kay 1995] Judy Kay. The um toolkit for cooperative user modelling. User Modeling and User-Adapted Interaction, vol. 4, no. 3 p.p. 149–196, 1995. `http://www.cs.usyd.edu.au/~judy/Homec/Pubs/um.ps` [Last access July 1st, 2005].

[Kay 2000a] Judy Kay. Stereotypes, Student Models and Scrutability. In Proceedings of the 5th International Conference on Intelligent Tutoring Systems (ITS 2000), P.p. 19–30, 2000.

[Kay 2000b] Judy Kay. User Interfaces for All, chapter User Modeling for Adaptation, P.p. 271–294. Human Factors Series. Lawrence Erlbaum Associates, Inc., 2000. `http://www.cs.usyd.edu.au/~judy/Homec/Pubs/ch18.pdf` [Last access June 1st, 2005].

[Kim 2002] Won Kim. Personalization: Definition, Status, and Challenges ahead. Journal of Object Technology, vol. 1, no. 1 p.p. 29–40, 2002. `http://www.jot.fm/issues/issue_2002_05/column3.pdf` [Last access May 25th, 2005].

[Kinshuk and Lin 2003] Kinshuk and Taiyu Lin. User Exploration Based Adaptation in Adaptive Learning Systems. International Journal of Information Systems in Education, vol. 1, no. 1 p.p. 22–31, 2003. `http://infosys.massey.ac.nz/~kinshuk/papers/jsise2002.pdf` [Last access May 19th, 2005].

[Kinshuk 1996] Kinshuk. Computer Aided Learning for Entry Level Accountancy Students. PhD thesis, De Montfort University, 1996. `http://infosys.massey.ac.nz/~kinshuk/thesis/mainpageps.html` [Last access June 25th, 2005].

[Kobsa and Pohl 1995] Alfred Kobsa and Wolfgang Pohl. The User Modeling Shell System BGP-MS. User Modeling and User-Adapted Interaction, vol. 4, no. 2 p.p. 59–106, 1995. `http://www.ics.uci.edu/~kobsa/papers/1995-UMUAI-kobsa.pdf` [Last access July 25th, 2005].

[Kobsa 1991] Alfred Kobsa. First experiences with the SB-ONE knowledge representation workbench in natural-language applications. ACM SIGART Bulletin, vol. 2, no. 3 p.p. 70–76, 1991.

[Kobsa 1993] Alfred Kobsa. User Modeling: Recent Work, Prospects and Hazards. In Adaptive User Interfaces: Principles and Practise. M. Schneider-Hufschmidt,

T. Kühme and U. Malinowski, eds., 1993. http://www.ics.uci.edu/~kobsa/papers/1993-aui-kobsa.pdf [Last access May 25th, 2005].

[Kobsa 2001a] Alfred Kobsa. Generic User Modeling Systems. User Modeling and User-Adapted Interaction, vol. 11, no. 1-2 p.p. 49–63, 2001. http://www.ics.uci.edu/%7Ekobsa/papers/2001-UMUAI-kobsa.pdf [Last access May 25th, 2005].

[Kobsa 2001b] Alfred Kobsa. Tailoring Privacy to User's Needs. In Proceedings of the 8th International Conference on User Modeling 2001 (UM2001), P.p. 303–313, 2001. http://www.ics.uci.edu/~kobsa/papers/2001-UM01-kobsa.pdf [Last access September 25th, 2005].

[Koch 2000] Nora Koch. Software Engineering for Adaptive Hypermedia Systems. PhD thesis, Ludwig-Maximilians-University Munich/Germany, 2000. http://www.pst.informatik.uni-muenchen.de/personen/kochn/PhDThesisNoraKoch.pdf [Last access May 8th, 2005].

[Konstan *et al.* 1997] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. GroupLens: applying collaborative filtering to Usenet news. Communications of the ACM, vol. 40, no. 3 p.p. 77–87, 1997.

[Kreger 2001] Heather Kreger. Web Services Conceptual Architecture (WSCA 1.0). Technical report, IBM Software Group, 2001. http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf [Last access August 28th, 2005].

[Kurhila *et al.* 2001] Jaakko Kurhila, Miikka Miettinen, Markku Niemivirta, Petri Nokelainen, Tomi Silander, and Henry Tirri. Bayesian Modeling in an Adaptive On-Line Questionnaire for Education and Educational Research. In Proceedings of the 10th International PEG Conference, P.p. 194–201, 2001. http://cosco.hiit.fi/Articles/peg2001eduform.pdf [Last access June 28th, 2005].

[Lane 2000] Carla Lane. Implementing Multiple Intelligences and Learning Styles in Distributed Learning/IMS Projects. Technical report, The Education Coalition (TEC), 2000. http://www.tecweb.org/styles/imslsindl.pdf [Last access June 10th, 2005].

[Li and Ji 2005] Xiangyang Li and Qiang Ji. Active Affective State Detection and Assistance with Dynamic Bayesian Networks. IEEE Transactions on Systems, Man, and Cybernetics: Special Issue on Ambient Intelligence, vol. 35, no. 1 p.p. 93–105, 2005.

[Mödritscher *et al.* 2004] Felix Mödritscher, Victor Manuel Garcia-Barrios, and Christian Gütl. The Past, the Present and the future of adaptive E-Learning. In Proceedings of the International Conference Interactive Computer Aided Learning (ICL2004), 2004. http://www.iicm.edu/iicm_papers/icl2004/adaptive_e-learning/adaptiv_e-learning.pdf [Last access April 28th, 2005].

[Murray 1987] Dianne M. Murray. Embedded User Models. In Proceedings of the 2nd IFIP International Conference on Human-Computer Interaction (INTERACT'87), P.p. 229–235, 1987.

[Olivier 2002] Bill Olivier. Standards and Specifications: Why Use IMS? Technical report, The Joint Information Systems Committee JISC, 2002. `http://www.jisc.ac.uk/uploaded_documents/bp4.pdf` [Last access February 13th, 2005].

[Openwings 2003] Openwings "Online Tutorial", 2003. `http://www.openwings.org/openwings-1.1/tutorial/index.html` [Last access March 30th, 2005].

[Oppermann *et al.* 1997] Reinhard Oppermann, Rossen Rashev, and Kinshuk. Knowledge Transfer (Vol. II), chapter Adaptability and Adaptivity in Learning Systems, P.p. 173–179. pAce, London, 1997. `http://fit.fraunhofer.de/%7Eoppermann/publications/kt97_gmd.pdf` [Last access April 20th, 2005].

[Orwant 1993] Jon Orwant. Doppelgänger Goes To School: Machine Learning for User Modeling. Master's thesis, Massachusetts Institue of Technology, 1993.

[Orwant 1995] Jon Orwant. Heterogeneous Learning in the Doppelgänger User Modeling System. User Modeling and User-Adapted Interaction, vol. 4, no. 2 p.p. 107–130, 1995. `ftp://ftp.media.mit.edu/pub/orwant/doppelganger/learning.ps.gz` [Last access June 21th, 2005].

[OSG 2003] OSGi Open Services Gateway Initiative. OSGi Service Platform Release 3, 2003. `http://www.osgi.org/osgi_technology/download_specs2.asp?section=2` [Last access September 1st, 2005].

[Oxford Advanced Learner's Dictionary 2005] Oxford Advanced Learner's Dictionary, 2005. `http://www.oup.com/elt/oald/` [Last access April 20th, 2005].

[Paramythis and Loidl-Reisinger 2003] Alexandros Paramythis and Susanne Loidl-Reisinger. Adaptive Learning Environments and e-Learning Standards. Electronic Journal of e-Learning, vol. 2, no. 11 p.p. 181–194, 2003. `http://www.ejel.org/volume-2/vol2-issue1/issue1-art11-paramythis.pdf` [Last access February 13th, 2005].

[Park and Lee 2003] Ok Park and Jung Lee. Handbook of Research for Educational Communications and Technology, chapter Adaptive Instructional Systems, P.p. 651–660. Association for Educational Communications and Technology, 2003. `http://coe.sdsu.edu/eet/articles/cmi/Park,%202003.pdf` [Last access May 18th, 2005].

[Razmerita *et al.* 2003] Liana Razmerita, Albert Angehrn, and Alexander Maedche. Ontology-based User Modeling for Knowledge Management Systems. In Proceedings of the 9th International Conference on User Modeling (UM'03), 2003. `http://www.calt.insead.edu/Project/OntoLogging/documents/2003-UM-Ontology_based_user_modeling_for_Knowledge_Management_Systems.pdf` [Last access July 21th, 2005].

[Rich 1979] Elaine Rich. User Modeling via Stereotypes. Cognitive Science, vol. 3, p.p. 329–354, 1979. http://www.cs.utexas.edu/users/ear/CogSci.pdf [Last access June 21th, 2005].

[RMI 1997] Sun Microsystems, Inc. Java Remote Method Invocation (RMI), Specification, 1997. http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html [Last access February 21th, 2005].

[Rodríguez-Estévez *et al.* 2003] Judith Rodríguez-Estévez, Manuel Caeiro-Rodríguez, and Juan M. Santos-Gago. Standardization in Computer Based Learning. The European Journal for the Informatics Professional, vol. 13, no. 3 p.p. 8–15, October 2003.

[Romand *et al.* 2005] Ed Romand, Gerald Brose, and Rima Patel Sriganesh. Mastering Enterprise JavaBeans. Wiley Publishing, Inc., 3rd edition, 2005. http://www.infosys.tuwien.ac.at/Staff/tom/Teaching/UniZH/CBSE/MasteringEJB3.pdf [Last access September 13th, 2005].

[Rousseau *et al.* 2004] Boris Rousseau, Parisch Browne, Paul Malone, and Mícheál ÓFoghlú. User Profiling for Content Personalisation in Information Retrieval. In Proceedings of the 19th ACM Symposium on Applied Computing (SAC '04), 2004. http://www.irisa.fr/texmex/publications/versionElect/2004/Rousseau_ACMSAC04_UserProfile.pdf [Last access October 13th, 2005].

[Russell 2003] Rosemary Russell. Metadata standards for the description of PORTAL users: a review. Technical report, UKOLN, University of Bath, 2003. http://www.fair-portal.hull.ac.uk/downloads/Metadata.pdf [Last access February 13th, 2005].

[Schreck 2001] Jörg Schreck. Security and Privacy in User Modeling. PhD thesis, University of Essen, 2001.

[Self 1987] John Self. Student models: What use are they? In P. Ercoli and R. Lewis, editors, Proceedings of the IEP TC3 Working Conference on AI Tools in Education, P.p. 73–86, 1987.

[Self 1993] John Self. Model-based Cognitive Diagnosis. User Modeling and User-Adapted Interaction, vol. 3, no. 2 p.p. 89–106, 1993. ftp://ftp.comp.lancs.ac.uk/pub/aai/aai-report-82.ps.Z [Last access June 13th, 2005].

[Self 1994] John Self. Student Modelling: the key to individualize knowledge-based instruction, chapter Formal Approaches to Student Modelling, P.p. 295–352. Springer-Verlag Berlin, 1994. ftp://ftp.comp.lancs.ac.uk/pub/aai/aai-report-92.ps.Z [Last access June 13th, 2005].

[Shehory and Sturm 2001] Onn Shehory and Arnon Sturm. Evaluation of modeling techniques for agent-based systems. In Proceedings of the 5th International Conference on Autonomous Agents (AGENTS '01), P.p. 624–631,

2001. http://www.sce.carleton.ca/faculty/esfandiari/agents/papers/shehory.pdf [Last access September 13th, 2005].

[Shute and Psotka 1996] Valerie J. Shute and Joseph Psotka. Handbook of Research on Educational Communications and Technology, chapter Intelligent tutoring systems: Past, Present and Future, P.p. 1–99. Scholastic Publications, 1996.

[Sison and Shimura 1998] Raymund Sison and Masamichi Shimura. Student Modeling and Machine Learning. International Journal of Artificial Intelligence in Education, vol. 9, p.p. 128–158, 1998.

[Smith *et al.* 2003] Michael Smith, Guy Bieber, and Jeff Carpenter. Openwings Management Service Specification Ver. 1.0 Final, 2003. http://www.openwings.org/download/specs/Openwings_Management_Services.pdf [Last access March 18th, 2005].

[Smythe *et al.* 2001] Colin Smythe, Frank Tansey, and Robby Robson. IMS Learner Information Package Information Model Specification, Version 1.0, 2001. http://www.imsglobal.org/profiles/ [Last access February 13th, 2005].

[Soller 2001] Amy L. Soller. Supporting Social Interaction in an Intelligent Collaborative Learning System. International Journal of Artificial Intelligence in Education, vol. 12, p.p. 40–62, 2001. http://www.ou.nl/otecresearch/publications/slavi%20stoyanov/stoyanov-kirschner.pdf [Last access May 8th, 2005].

[Stoyanov and Kirschner 2004] Slavi Stoyanov and Paul Kirschner. Expert Concept Mapping Method for Defining the Characteristics of Adaptive E-Learning: ALFANET Project Case. Educational Technology, Research & Developement, vol. 52, no. 2 p.p. 41–56, 2004. http://www.ou.nl/otecresearch/publications/slavi%20stoyanov/stoyanov-kirschner.pdf [Last access May 8th, 2005].

[Stratakis *et al.* 2003] Miltos Stratakis, Vassilis Christophides, Kevin Keenoy, and Aimilia Magkanaraki. SeLeNe - Preliminary Report: Learning Objects, Meta-Data and Standards. Technical report, School of Computer Science and Information Systems, 2003. http://www.dcs.bbk.ac.uk/selene/reports/KK_Preliminary_report.pdf [Last access March 2nd, 2005].

[Tsai and Machado 2002] Susanna Tsai and Paulo Machado. Essay: E-learning, online learning, web-based learning, or distance learning: unveiling the ambiguity in current terminology. eLearn, vol. 2002, no. 7 p.p. 3, 2002. http://www.elearnmag.org/subpage/sub_page.cfm?section=3\&list_item=6\&page=1 [Last access April 20th, 2005].

[Tsalgatidou and Pilioura 2002] Aphrodite Tsalgatidou and Thomi Pilioura. An Overview of Standards and Related Technology in Web Services. Distributed and Parallel Databases, vol. 12, p.p. 135–162, 2002. http://www.csd.uch.gr/~hy565/Papers/overview_of_standards.pdf [Last access July 18th, 2005].

[Tsiriga and Virvou 2003] Victoria Tsiriga and Maria Virvou. Initializing Student Models in Web-Based ITSs: A Generic Approach. In Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies (ICALT 2003), P.p. 42–46, 2003. `http://thalis.cs.unipi.gr/~vtsir/Tsiriga%40Virvou_ICALT2003.pdf` [Last access June 18th, 2005].

[ULF 2000] Saba Software Inc. Universal Learning Format (ULF) Technical Specification Version 1.0, 2000. `http://xml.coverpages.org/ulfSpecification20001204.pdf` [Last access February 13th, 2005].

[Vergara 1994] Harald Vergara. PROTUM: A Prolog Based Tool for User Modeling. Technical report, Department of Information Science, University of Konstanz, 1994.

[Wade *et al.* 2002] Vincent Wade, Kevin Riley, Bob Banks, Paul Foster, Neil Evans-Mudie, Yves Nicol, and Paul Doherty. Work Package 5 - Object (Interfaces) Specification. Technical report, GESTALT - Project AC367, 2002. `http://www.fdgroup.co.uk/gestalt/D502v4.zip` [Last access February 13th, 2005].

[Webb and Kuzmycz 1998] Geoffrey I. Webb and Mark Kuzmycz. Evaluation of Data Aging: A Technique for Discounting Old Data During Student Modeling. In Proceedings of the 4th International Conference on Intelligent Tutoring Systems (ITS'98), P.p. 384–393, 1998.

[Webb *et al.* 2001] Geoffrey I. Webb, Michael J. Pazzani, and Daniel Billsus. Machine Learning for User Modeling. User Models User-Adapted Interaction, vol. 11, no. 1-2 p.p. 19–29, 2001.

[Weibelzahl 2003] Stephan Weibelzahl. Evaluation of Adaptive Systems. PhD thesis, University of Trier, 2003. `http://www.easy-hub.org:8000/stephan/weibelzahl03-diss.pdf` [Last access May 8th, 2005].

[Westin 1970] Alan F. Westin. Privacy and Freedom. Bodley Head, 1970.

[Whatis.com 2005] Whatis.com. Whatis.com Website, 2005. `http://whatis.techtarget.com/` [Last access October 13th, 2005].

[Wilson and Jones 2002] Scott Wilson and Peter Rees Jones. What Is... IMS Learner Information Packaging?, 2002. `http://www.cetis.ac.uk/groups/20010801124300/FR20021029103504` [Last access February 13th, 2005].

[Zhou and Evens 1999] Yujian Zhou and Martha W. Evens. A Practical Student Model in an Intelligent Tutoring System. In Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'99), P.p. 13–18, 1999.