Universität Duisburg-Essen, Standort Essen
Fachbereich 6 Mathematik

User Modeling Servers — Requirements, Design, and Evaluation

Dissertation vorgelegt zum Erwerb
des akademischen Grades Dr. rer. nat.

von Josef Fink
aus Sigmaringen

Datum der mündlichen Prüfung: 15. Juli 2003
Gutachter: Prof. Dr. Alfred Kobsa, Prof. Dr. Rainer Unland

Acknowledgements

This thesis originated from research in several scientific environments. Basic ideas have been developed at the Universities of Konstanz and Essen within the BGP-MS project, which was funded by the German Science Foundation (DFG). Most of the work has been carried out at GMD, German National Research Center for Information Technology, within the Deep Map project, which was funded by the European Media Lab (EML), Heidelberg, Germany. The multi-disciplinary perspective to user modeling and user-adaptive systems pursued in this thesis follows the approach taken in several projects at GMD including the AVANTI and the HIPS project, which were partially funded by the European Commission, and the LaboUr project, which was funded by the German Science Foundation (DFG). Important parts of the implementation and evaluation work have been carried out at humanIT, Human Information Technologies AG, Sankt Augustin, Germany.

It was Professor Alfred Kobsa, who attracted me to the field of user modeling and to this thesis project many years ago. Since that time, he carefully guided me through this long-term project even in times when I was seemingly absent. For this long-lasting mentorship, I am especially indebted to him. I would also like to thank Professor Rainer Unland, who swiftly accepted to act as a second supervisor and provided valuable comments on earlier versions of this dissertation.

During my affiliation with the aforementioned scientific and commercial institutions, I was privileged to work with and learn from many colleagues and friends; the following list does not mention all of them: Lucian Ghitun, Jörg Höhle, Viorel Holban, Jürgen Koenemann, Detlef Küpper, Hans-Günter Lindner, Rainer Malaka, Andreas Nill, Stephan Noller, Reinhard Oppermann, Wolfgang Pohl, Jörg Schreck, and Ingo Schwab. Thanks to all of them for sharing their talents with me!

In particular, I would like to thank Jürgen Koenemann, who helped me articulating and pursuing a clear thesis statement and pointed out many shortcomings in earlier versions of this dissertation. Any weaknesses, misunderstandings, and errors are, however, solely my responsibility.

Apart from science, getting a dissertation project done is also a matter of focus, persistence, and sedulity. In this vein, I would like to thank Christoph Thomas and Erich Vorwerk, who, despite of my resistance, never stopped carefully pushing me forward.

Finally, I thank the six most important people in my life for their unlimited support in all respects. Without their assistance, patience, and love, I would have probably never completed this project. In regard to their unique contribution, I dedicate this work to them.

Foreword

Preliminary versions of Chapter 2.2, Chapter 3, Chapters 7 and 8, and Chapter 9 of this thesis have been partially published in Fink [1999], Fink and Kobsa [2000], Fink and Kobsa [2002], and Kobsa and Fink [2003].

Zusammenfassung

Softwaresysteme, die ihre Services an Charakteristika individueller Benutzer anpassen (beispielsweise an Interessen, Präferenzen, Erfahrung und Wissen) haben sich bereits als effektiver und/oder benutzerfreundlicher als statische Systeme in mehreren Anwendungsdomänen erwiesen. Beispiele für solche individuellen Anpassungen sind auf die Benutzerexpertise zugeschnittene Hilfetexte, angemessene Produktpräsentationen in elektronischen Medien, gefilterte Ergebnisse aus Suchmaschinen im World Wide Web, pro-aktiv angebotene Tipps zur Erweiterung von Benutzerfähigkeiten, Hinweise auf potenziell relevante Nachrichten beziehungsweise Produkte und Empfehlungen bezüglich individueller Lernstrategien. Um solche Anpassungsleistungen anbieten zu können, greifen benutzeradaptive Systeme auf Modelle von Benutzercharakteristika zurück. Der Aufbau und die Verwaltung dieser Modelle wird durch dezidierte Benutzermodellierungskomponenten vorgenommen.

Ein wichtiger Zweig der Benutzermodellierungsforschung beschäftigt sich mit der Entwicklung sogenannter ‚Benutzermodellierungs-Shells', d.h. generischen Benutzermodellierungssystemen, die die Entwicklung anwendungsspezifischer Benutzermodellierungskomponenten erleichtern. Die Bestimmung des Leistungsumfangs dieser generischen Benutzermodellierungssysteme und deren Dienste bzw. Funktionalitäten wurde bisher in den meisten Fällen intuitiv vorgenommen und/oder aus Beschreibungen benutzeradaptiver Systeme in der wissenschaftlichen Literatur abgeleitet. Wegen der hohen Affinität der Benutzermodellierungsforschung zum Forschungsgebiet der Künstlichen Intelligenz wurde die Benutzermodellierung vornehmlich als ein Prozeß der Wissensverarbeitung angesehen. Die in verwandten Gebieten wie Datenbank- und Transaktionsmanagement, verteilte Informationssysteme, Informationswissenschaft und Wirtschaftsinformatik reichlich vorhandenen Erfahrungen bez. Entwurf, Implementierung und Einsatz von Server-Technologie wurden nicht in Betracht gezogen. Die meisten Benutzermodellierungs-Shells fanden (vielleicht aus den vorgenannten Gründen?) keine nennenswerte Verbreitung, nur wenige verließen die Forschungseinrichtungen, an denen sie ursprünglich entwickelt wurden.

In der jüngeren Vergangenheit führte der Trend zur Personalisierung im World Wide Web zur Entwicklung mehrerer kommerzieller Benutzermodellierungsserver. Die für diese Systeme als wichtig erachteten Eigenschaften stehen im krassen Gegensatz zu denen, die bei der Entwicklung der Benutzermodellierungs-Shells im Vordergrund standen und umgekehrt. Kommerzielle Benutzermodellierungsserver weisen Eigenschaften auf, die für deren Einsatz in realen Anwendungsumgebungen von essentieller Bedeutung sind, beispielsweise die Integration extern vorhandener Benutzerinformationen, Repräsentation von Benutzerverhalten, Skalierbarkeit im Hinblick auf eine wachsende Anzahl von Benutzern und Unterstützung des Datenschutzes. Schon eine oberflächliche Analyse zeigt jedoch auch bei diesen Systemen noch ein erhebliches Verbesserungspotenzial auf, beispielsweise bezüglich (i) der eingesetzten Lerntechniken mit einem Schwerpunkt auf der Integration von Domänenwissen und der Kombination von Lerntechniken zum Einsatzzeitpunkt, (ii) Erweiterbarkeit und (iii) der Integration extern vorhandener Benutzerinformationen. Angesichts dieser komplementären Vor- und Nachteile kommerzieller Benutzermodellierungsserver verwundert es, dass diese Systeme offenbar noch keinen Eingang in die Benutzermodellierungsliteratur gefunden haben.

Vor diesem Hintergrund ist das Ziel dieser Dissertation (i) Anforderungen an Benutzermodellierungsserver aus einer *multi-disziplinären wissenschaftlichen* und einer einsatzorientierten (kommerziellen) *Perspektive* zu analysieren, (ii) einen Server zu entwerfen und zu implementieren, der diesen Anforderungen genügt, und (iii) die Performanz und Skalierbarkeit dieses Servers unter der Arbeitslast kleinerer und mittlerer Einsatzumgebungen gegen die diesbezüglichen Anforderungen zu überprüfen.

Um dieses Ziel zu erreichen, verfolgen wir einen anforderungszentrierten Ansatz, der auf Erfahrungen aus verschiedenen Forschungsbereichen, insbesondere Benutzermodellierung, benutzeradaptiven Systemen, Datenbank- und Transaktionsmanagement sowie Marketingforschung aufbaut. Wir entwickeln zwei Anforderungskataloge, einen für eher generelle Anforderungen an Server (wie beispielsweise Mehrbenutzersynchronisation, Transaktionsmanagement und Zugriffskontrolle) und einen zweiten für Anforderungen, die spezifisch für die Benutzermodellierung sind (wie beispielsweise Funktionalität, Datenakquisition, Erweiterbarkeit und Flexibilität, Integrierbarkeit externer Benutzerinformationen, Kompatibilität zu Standards und Unterstützung des Datenschutzes). Auf Basis des zweiten Katalogs besprechen und vergleichen wir in der Folge ausgewählte kommerzielle Benutzermodellierungsserver. Eine vergleichbare Analyse wurde bisher im Forschungsbereich Benutzermodellierung nicht durchgeführt.

Gestützt auf die beiden Anforderungskataloge entwickeln wir dann eine generische Architektur für einen Benutzermodellierungsserver, die aus einem Serverkern für das Datenmanagement und modular hinzufügbaren Benutzermodellierungskomponenten besteht, von denen jede eine wichtige Benutzermodellierungstechnik implementiert. Um einen geeigneten Serverkern zu finden, vergleichen und evaluieren wir in der Folge gängige Verzeichnis- und Datenbankmanagementsysteme. Dabei beziehen wir nicht nur heutige, sondern auch zukünftige Benutzermodellierungsszenarien in unsere Betrachtung mit ein. Als Ergebnis kommen wir zu dem Schluss, dass Verzeichnisdienste (die bisher noch nicht als Basis für Benutzermodellierungsserver eingesetzt wurden) generell Datenbankmanagementsystemen überlegen sind, beispielsweise im Hinblick auf Flexibilität und Erweiterbarkeit, Verwaltung verteilter Informationen, Replikationsgrad, Performanz, Skalierbarkeit und Kompatibilität zu Standards.

Um die Zweckmäßigkeit unserer generischen Serverarchitektur nachzuweisen, beschreiben wir in der Folge den Benutzermodellierungsserver, den wir für ‚Deep Map' entwickelt haben, einem Projekt, das sich mit der Entwicklung eines portablen benutzeradaptiven Touristenführers beschäftigt. Wir beschreiben die Benutzermodellierungskomponenten, die wir für dieses Einsatzszenario entwickelt haben, und insbesondere die in diesen Komponenten enthaltenen Lerntechniken, die wir aus dem Bereich des Maschinellen Lernens für Benutzermodellierung übernommen haben. Wir zeigen, dass wir durch die Integration dieser Benutzermodellierungskomponenten in einem Server Synergieeffekte zwischen den eingesetzten Lerntechniken erzielen und bekannte Defizite einzelner Verfahren kompensieren können, beispielsweise bezüglich Performanz, Skalierbarkeit, Integration von Domänenwissen, Datenmangel und Kaltstart.

Abschließend präsentieren wir die wichtigsten Ergebnisse der Experimente, die wir durchgeführt haben um empirisch nachzuweisen, dass der von uns entwickelte Benutzermodellierungsserver den zentralen Performanz- und Skalierbarkeitskriterien unserer

Anforderungskataloge genügt. Wir beginnen mit einer Beschreibung unseres Testansatzes und der empirisch überprüften Arbeitslast, die wir in Anlehnung an reale Einsatzbedingungen simuliert haben. Wir präsentieren ausgewählte Ergebnisse unserer Experimente und diskutieren Stärken und Schwächen unseres Benutzermodellierungsservers. Als Hauptergebnis stellen wir fest, dass unser Benutzermodellierungsserver die vorbesagten Kriterien in Anwendungsumgebungen mit kleiner und mittlerer Arbeitslast in vollem Umfang erfüllt. Die Verarbeitungszeiten für eine repräsentativ zusammengestellte Menge an Benutzermodellierungsoperationen wachsen nur degressiv mit der Häufigkeit der Seitenanfragen. Die Verteilung des Benutzermodellierungsservers auf mehrere Rechner beschleunigte zusätzlich die Verarbeitung derjenigen Operationen, die parallel ausgeführt werden können. Ein Test in einer Anwendungsumgebung mit mehreren Millionen Benutzerprofilen und einer Arbeitslast, die als repräsentativ für größere Web Sites angesehen werden kann bestätigte, dass die Performanz der Benutzermodellierung unseres Servers keine signifikante Mehrbelastung für eine personalisierte Web Site darstellt. Gleichzeitig können die Anforderungen unseres Benutzermodellierungsservers an die verfügbare Hardware als moderat eingestuft werden. Eine vergleichbare Untersuchung wurde bisher in der Benutzermodellierungsforschung nicht durchgeführt.

Wir erwarten, dass unsere Arbeit den Entwurf, die Implementierung und den Einsatz benutzermodellierender und –adaptiver Systeme sowohl in Forschungs-, als auch in kommerziellen Umgebungen beeinflussen wird. Unser Benutzermodellierungsserver wird in kommerziellen Anwendungsumgebungen mit mehreren Millionen Benutzern bereits erfolgreich eingesetzt.

Abstract

Software systems that adapt their services to characteristics of individual users (e.g., their interests, preferences, proficiencies and knowledge) have already proven to be more effective and/or usable than non-adaptive systems in several application domains. Individualized tailoring has been used to, e.g., cater help text to the user's level of expertise, chose appropriate product presentations in electronic offerings, filter retrieval results of Web search engines, provide unsolicited tips to extend the user's skills set, present news flashes or product recommendations to users in which they are probably interested, and recommend personalized learning strategies. For exhibiting such personalized behavior, user-adaptive software systems rely on models of user characteristics. Acquisition and management of these models is carried out by dedicated user modeling components.

An important strand of user modeling research is devoted to developing so-called 'user modeling shell systems', i.e. generic user modeling systems that facilitate the development of application-specific user modeling components. The decisions as to what these generic user modeling systems and their respective services/functionalities are were mostly based on intuition and/or experience gained from studying user-adaptive applications as reported in the scientific literature. Due to the strong affinity of user modeling research to artificial intelligence in these days, user modeling was mainly considered a knowledge processing task. The rich experience that related research areas like database and transaction management, distributed information systems, information science, and management information systems had acquired regarding the design, implementation and deployment of server technology was not taken into account. Most of these user modeling shell systems (therefore?) did not enjoy much distribution; only a few ever left the research institutions where they were originally developed.

More recently, the trend towards personalization on the World Wide Web led to the development of several commercial user modeling servers. Features that are deemed to be important for these systems contrast sharply with those regarded as important for user modeling shell systems, and vice versa. Commercial user modeling servers exhibit deployment-supporting characteristics that are of paramount importance in real-world environments, including integration of external user-related information, representation of user behavior, scalability in terms of an increasing number of users, and support for user privacy. However, even a superficial analysis reveals that these commercial systems are lacking as well, e.g. with regard to (i) learning techniques focused on the integration of domain knowledge and technique mix at deployment time, (ii) extensibility, and (iii) integration of user-related information that is external to the user modeling server. Given these complementary strengths and weaknesses of commercial user modeling servers, it is surprising that most of them seemingly have not even been mentioned in the user modeling literature.

Against this background, the aim of this dissertation is to (i) analyze the requirements that user modeling servers must meet to be acceptable both from a *multi-disciplinary scientific perspective* and from the viewpoint of (commercial) deployment, (ii) design and implement a server that meets these requirements, and (iii) verify its compliance with core performance and scalability requirements under the workload of small and medium-sized real-world environments.

In order to achieve this, we follow a requirements-driven approach, thereby drawing on experience from a variety of research areas including user modeling, user-adaptive systems, database and transaction management, management information systems, and marketing research. We develop two requirements catalogues, one for more general server requirements (e.g., multi-user synchronization, transaction management, and access control) and the other for requirements that are specific to user modeling (e.g., functionality, data acquisition, extensibility and flexibility, integration of external user-related information, compliance with standards, and support for privacy). Based on the latter, we conduct a review of selected commercial user modeling servers and compare and discuss our findings. A comparable analysis has not been conducted so far in user modeling research.

Based on these requirement catalogues, we develop a generic architecture for a user modeling server that consists of a server core for data management and several 'pluggable' user modeling components, each of which implements an important user modeling technique. In order to determine an appropriate server core, we compare and evaluate common directory and database management systems. We thereby not only take current, but also future user modeling scenarios into account. We find that directory management systems (which have never been used before as a basis for user modeling servers) are generally superior to database management systems with regard to, e.g., flexibility and extensibility, management of distributed information, replication scale, performance, scalability, and compliance with standards.

To prove the validity of our generic server architecture, we subsequently describe the user modeling server that we developed for 'Deep Map', a project that is concerned with the construction of a portable user-adaptive tourist guide. We describe the user modeling components that we developed for this specific deployment scenario, and specifically the incorporated learning techniques that we adopted from the area of machine learning for user modeling. We argue that by integrating the user modeling components into a single server, we can leverage several synergistic effects between these techniques and compensate for well-known deficits of individual techniques with regard to, e.g., performance, scalability, integration of domain knowledge, sparsity of data, and cold start.

Finally, we present the most important results of the experiments that we conducted to empirically verify the compliance of our user modeling server with core performance and scalability requirements introduced earlier in our requirement catalogues. We start with a brief description of our testing approach and the empirically verified real-world workload that we simulated. We present selected results and discuss strengths and weaknesses of our server. As a main result, we argue that our user modeling server can fully cope with small and medium-sized application workloads. The processing time for a representative mix of user modeling operations was found to only degressively increase with the frequency of page requests. The distribution of the user modeling server across a network of computers additionally accelerated those operations that are amenable to parallel execution. A large-scale test with several million user profiles and a page request rate that is representative of major Web sites confirmed that the user modeling performance of our server will not impose a significant overhead for a personalized Web site. At the same time, the hardware demands of our user modeling server are moderate. A comparable evaluation has not been carried out in user modeling research so far.

We expect that our work impacts the design, implementation, and deployment of user modeling and user-adaptive systems both in research and commercial environments. Our user modeling server has already been successfully deployed to commercial application environments with several millions of users.

Contents

List of Figures

List of Tables

1 Introduction

1.1 History of User Modeling Servers

Over the past two decades, a plethora of user-adaptive application systems have been developed in user modeling research that acquire and maintain relevant information about their users, and provide different kinds of adaptation to them (for an overview and discussion of several systems, we refer to Kobsa and Wahlster [1989], McTear [1993], Brusilovsky [1996], Brusilovsky et al. [1998], Jameson [1999], and Kobsa et al. [2001]). In most of these systems, however, user modeling functionality was an integral part of the user-adaptive application. This 'monolithic' approach hampered employment of user-related information in several user-adaptive applications as well as the reuse of user modeling functionality in further user modeling systems.

In the late eighties and early nineties, a parallel strand of user modeling research aimed at overcoming these limitations by developing so-called 'user modeling shell systems' (see Kobsa and Pohl [1995; 1998] and Kobsa [2001a]), i.e. generic user modeling systems that facilitate the development of application-specific user modeling components. The decisions as to what these generic user modeling components and their respective services/functionalities are were mostly based on intuition and/or experience gained from studying (the literature of a few) user-adaptive applications (cf. Kobsa [2001a]). Due to the strong affinity of user modeling research in these days especially to artificial intelligence, user modeling has been mainly considered a knowledge processing task (e.g., in Pohl [1998]). Consequently, important features of systems like 'UMT' [Brajnik and Tasso, 1994], 'BGP-MS' [Kobsa and Pohl, 1995; Pohl, 1998], 'Doppelgänger' [Orwant, 1995], and 'TAGUS' [Paiva and Self, 1995] include

- *generality including domain independence* (except for domain-dependent systems in the area of adaptive tutoring like TAGUS),
- *expressiveness* (i.e., maintaining as many types of assumptions about users' propositional attitudes[1] as possible), and especially
- *strong representational and inferential capabilities* (e.g., reasoning in first-order predicate logic, modal reasoning, reasoning with uncertainty).

Some of the aforementioned user modeling shells included some server features (e.g., BGP-MS [Kobsa and Pohl, 1995; Pohl and Höhle, 1997; Pohl, 1998; Schreck, 2003] and Doppelgänger). But again, the development of these features was mostly driven by intuition and not by requirements that have been elicited from studying needs of several user-adaptive applications. And to the best of our knowledge, the rich experience in server design, implementation, and deployment from related research areas like database and transaction management, distributed systems design, information science, and management

[1] Propositional attitudes are for example a user's interests and preferences, knowledge, beliefs, and goals. Only a few shell systems aimed at modeling users' behavior as well, e.g. when reading articles in an electronic newspaper [Orwant, 1995].

information systems was not taken into account (see Fink [1996; 1999] for notable exceptions).

Most of the aforementioned user modeling shell systems did not enjoy much distribution. A notable exception seems to be BGP-MS, which was used at a few research sites outside of the institutions at which it was originally developed, and especially ‘GroupLens’ [Resnick et al., 1994; Konstan et al., 1997; Net Perceptions, 2000], which turned into a commercial product in the late nineties (see Chapter 3.2.1).

In parallel to these research efforts, the trend towards personalization[2] on the World Wide Web led to the development of several commercial user modeling servers. The rationale behind their development was to support companies in developing and deploying user-adaptive Web sites. Features that are regarded as important for commercial user modeling servers contrast sharply with those regarded as important for user modeling shell systems, and vice versa. In general, commercial user modeling servers focus on deployment-supporting features that seem to be of paramount importance in real-world environments like *integration of external user-related information*, *behavior-oriented representation*, *scalability in terms of an increasing number of users*, and *support for user privacy*. Given these complementary strengths and weaknesses of commercial user modeling servers it is surprising that most commercial user modeling servers seem to be not even mentioned in the user modeling literature (a notable exception seems to be Fink and Kobsa [2000]). Based on this, we expect that a presentation and discussion of these systems and their features provides a valuable source of requirements for the development of our user modeling server as well as a source of information and inspiration for further user modeling research.

Presenting and discussing these user modeling servers seems hardly appropriate, however, without analyzing the rationale behind their design and deployment. Especially experience from marketing research and practice seems to have considerably shaped these systems and motivates their deployment. In the following sub-chapter we therefore present a brief overview of personalization in e-commerce, including customer relationship management. Thereby, we also lay the basis for the requirements analysis we conduct in the first part of our work.

1.2 Personalization in E-Commerce

In several application domains, user-adaptive software systems have already proven to be more effective and/or usable than non-adaptive systems. One of these classes of adaptive systems with clear user benefits are user-adaptive tutoring systems which were shown to often significantly improve the overall learning progress. These systems and their benefits have already been extensively reviewed in the user modeling literature (see e.g. most of the papers in Brusilovsky et al. [1998] and the evaluations in Eklund and Brusilovsky [1998]; moreover, see Specht [1998] and Specht and Kobsa [1999]).

[2] In e-commerce, ‘personalization’ is used as a generic term that denotes user-adaptive system features and user modeling issues as well. Despite its ambiguity, we will employ this term throughout this thesis. In cases where it is necessary to refer to one of the two meanings, we will use well-established and more specific terms from user modeling research like ‘adaptivity’ and ‘user modeling’.

Less represented in the user modeling literature are user-adaptive (aka ‘personalized’) systems for e-commerce including customer relationship management. A few notable exceptions are Popp and Lödel [1996], Åberg and Shahmehri [1999], Ardissono and Goy [1999; 2000], and Jörding [1999]. This is surprising since there already exists ample evidence for personalization going mainstream in e-commerce. According to Manna [2000a], Appian estimates that the revenues made by the online personalization industry, including custom development and independent consulting, will reach $1.3 billion in 2000, and $5.3 billion by 2003 [Appian, 2000a]. Ovum forecasts that the world-wide revenues for personalization software will rise from $10.85 million in 2000 to $93.4 million in 2005 (see Figure 1-1) [Millhouse et al., 2000]. Gartner predicts that “by 2003, nearly 85 percent of global 1,000 Web sites will use some form of personalization (0.7 probability)”[3] [Abrams et al., 1999]. There are also many indications that personalization provides substantial benefits in this application domain as well [Hof et al., 1998; Bachem, 1999; Cooperstein et al., 1999; Hagen et al., 1999; Kobsa et al., 2001].

Utilizing personalization and the underlying ‘one-to-one’ marketing paradigm is of paramount importance for businesses in order to be successful in today’s short-lived, complex, and highly competitive markets [Peppers and Rogers, 1993; 1997; Allen et al., 1998]. One-to-one builds on the basic principles of knowing and remembering a customer and serving him as an individual. From a marketing point of view, traditional communication channels between a company and its customers continuously decrease in efficiency due to market saturation, product variety, and increasingly complex and autonomous behavior of clients with respect to goods (e.g., drivers of luxury cars can at the same time be regular customers at discount shops) and media (e.g., people use different media like television, newspapers and the Internet, sometimes even in parallel) [Bachem, 1999]. Against this background, traditional user segmentations in marketing research with their inherent simplicity (e.g., customer behavior can be predicted from a few key characteristics), linearity (i.e., future customer behavior can be predicted from past behavior), and time invariance (i.e., market rules always apply) provide less and less useful information for adequate personalization and have to be complemented by the latest information about customers directly elicited from their (on-line) behaviors. Thereby, marketers expect to get more insights into the many facets of customer behavior which is often fairly complex, non-linear, and time-variant [Bachem, 1999; Cooperstein et al., 1999].

[3] This follows the ranking of the world’s best performing companies that is annually carried out by Business Week [1999].

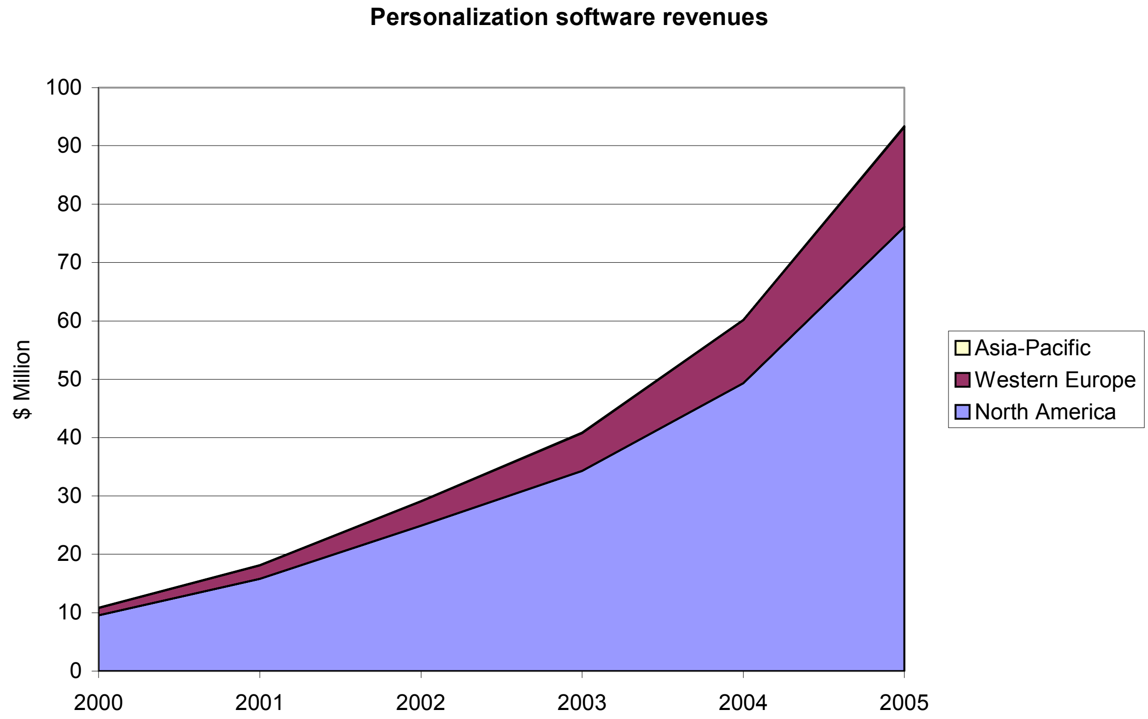


Figure 1-1: Personalization software revenues (based on Millhouse et al. [2000])[4]

Forrester Research reports regularly about the personalization activities of selected e-commerce sites, for example in Hagen et al. [1999] about the efforts and resulting benefits of 54 U.S. sites. Figure 1-2 depicts some examples of personalized information and services these sites offer to their users. Allen et al. [1998] describe 29 personalized Web sites. Schafer et al. [1999] reviews the personalized Web services and associated benefits of well-known e-commerce companies like Amazon.com, CDnow, eBay, Levis, E! Online, and Reel.com.

In general, personalization has been reported to provide benefits throughout the customer life cycle including drawing new visitors, turning visitors into buyers, increasing revenues, increasing advertising efficiency, and improving customer retention rate and brand loyalty [Hof et al., 1998; Bachem, 1999; Cooperstein et al., 1999; Hagen et al., 1999; Schafer et al., 1999]. Jupiter Communications reports that personalization at 25 consumer e-commerce sites increased the number of new customers by 47% in the first year, and revenues by 52% [Hof et al., 1998]. 'Nielsen//NetRatings' [ICONOCAST, 1999] report that e-commerce sites offering personalized services convert significantly more visitors into buyers than e-commerce sites that do not offer personalized services. Although the research approach taken is not always transparent and/or satisfactory (e.g., regarding the methodology used and the conclusions drawn[5]), these figures indicate that personalization offers at least in part

[4] The forecasts for Asia-Pacific are very small (i.e., from $0.02 million in 2000 to $0.21 million in 2005) and therefore hardly visible.

[5] One problem for instance is that personalization is hardly ever introduced in isolation on a Web site, but in most cases together with other company measures that may also have an effect on the addressed benefits (e.g., marketing

significant benefits[6]. Besides this evidence, there seems to be an even greater potential for personalization improving customer retention and brand loyalty. According to Peppers and Rogers [1993] and Reichheld [1996], improving customer retention and brand loyalty directly leads to increased profits because it is much cheaper to sell to existing customers than to acquire new ones (since the costs of selling to existing customers decrease over time and since the spending of loyal customers tends to accelerate and increase over time). Consequently, businesses today focus on retaining those customers with the highest customer life time value, on developing those customers with the most unrealized strategic life time value, and on realizing these profits with each customer individually [Cooperstein et al., 1999; Peppers et al., 1999].

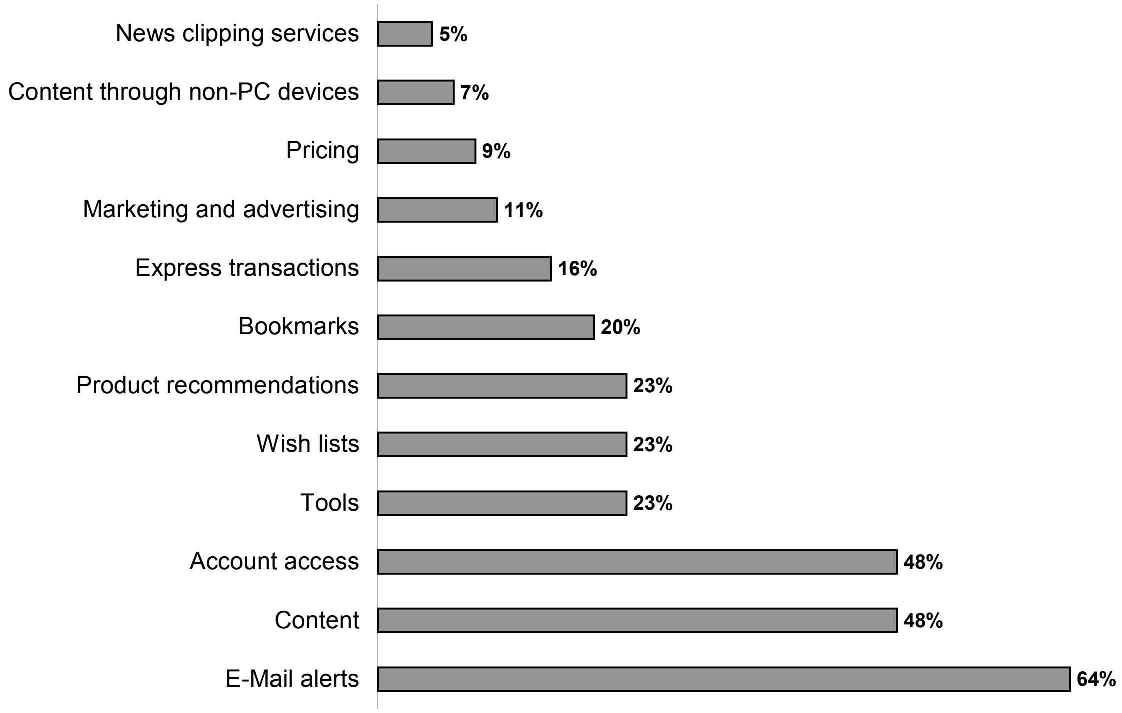


Figure 1-2: Commercial personalization examples (based on Hagen et al. [1999])

In parallel to the advent of personalized e-commerce sites, numerous tool systems emerged during the last few years that aim at assisting companies in developing and deploying personalized Web sites. As opposed to many academic user modeling systems, nearly all of them have been developed as server systems right from the beginning. We believe that the development of these server systems has been mainly motivated by the substantial benefits

and promotion measures, improved customer service, improved site navigation, and reduced response times [Cooperstein et al., 1999]).

[6] Despite of these success figures, there is also evidence for poorly done personalization leading to lower customer retention, reduced profit margins, and lost sales [Hagen et al., 1999]. The authors found a personalized drugstore that allows users to disclose allergy information, but recommended a drug that was unsuitable for people with the allergy that the user had entered. Affected users are likely to leave this Web shop, possibly forever. Another example is a Web store that presented an advertisement for a $19.95 surge protector to a user who had already put a $59.95 model in her shopping cart.

of centralized user modeling. In the following sub-chapter, we substantiate these potential benefits, thereby taking advantage of experience that motivated more than a decade ago the shift towards centralized data management (e.g., Martin [1983], Zehnder [1985], Date [1986]). Subsequently, we argue that centralized user modeling is one extreme within a continuum of potential distribution schemes and that current and especially future user modeling scenarios probably require for an architecture that comprises (elements of) both centralized and decentralized user modeling (systems).

1.3 Centralized vs. Decentralized User Modeling

For exhibiting personalized behavior, software systems rely on a model of relevant user characteristics (e.g., interests, preferences, proficiencies, knowledge). Acquisition and management of these models is carried out by a dedicated user modeling component. Most of the research prototypes that have been developed so far follow a monolithic approach with the user modeling component being embedded in and becoming an integral part of the user-adaptive application (see for example Finin [1989], Brajnik and Tasso [1994], Kay [1995], and Weber and Specht [1997]). A parallel strand of research focused on centralized autonomous user modeling[7] and led to the development of a comparatively small number of user modeling servers (see for example Kobsa and Pohl [1995], Orwant [1995], Konstan et al. [1997], Machado et al. [1999], and Billsus and Pazzani [2000]). In contrast with this, most current commercial user modeling systems have been designed as server systems right from the beginning (a notable exception from this is 'Open Sesame!' [Caglayan et al., 1997], an interface agent that maintains all information about the user in an embedded user modeling component[8]).

Compared to embedded user modeling systems, user modeling servers seem to provide promising advantages regarding their deployment, including the following ones (see also Billsus and Pazzani [2000]):

- *Up-to-date user information for holistic personalization.* Information about the user, her system usage, and the usage environment is maintained by a (central) user modeling server and put at the disposal of more than one application at the same time. Such a central repository of user information is in sharp contrast with the scattered and partially redundant modeling of user characteristics within today's applications (including those on the World Wide Web). One can assume that from a user's point of view, such a central repository will significantly contribute to a more consistent and coherent working environment comprising different user-adaptive applications.
- *Synergistic effects with respect to acquisition and usage*. User information acquired by one application can be employed by other applications and vice versa. Examples for such a scenario are different types of news readers [Resnick et al., 1994]; news readers and personalized agents [Good et al., 1999]; and various sensor applications, an e-mail

[7] Centralized user modeling does not necessarily imply physical centralization of user-related information (although this has been the case in all research prototypes developed so far). A promising alternative seems to be the concept of virtually centralized user information (see Chapter 3.3).

[8] The reason for this 'abnormality' seems to be that Open Sesame! was originally released as a desktop learning agent (i.e., a user-adaptive application that incorporates user modeling functionality). More recently, the development of Open Sesame! has been abandoned in favor of the user modeling server Learn Sesame [Caglayan et al., 1997; Open Sesame, 2000]. For more information on Open Sesame! and Learn Sesame, we refer to Chapter 3.2.4.

filtering application and a personalized newspaper [Orwant, 1995]. Acquisition and representation components of a user modeling server can be expected to take advantage of synergistic effects as well [Pohl and Nick, 1999].

- *Low redundancy with respect to application and domain independent information*. Information about, e.g. users' competence in handling computers, like the ability to manipulate interface elements within a WIMP (Windows, Icons, Menus, Pointer) interface, can be stored with low redundancy in a user modeling server [Fink et al., 1998] to make it available to all applications which they use.
- *Low redundancy* with respect to *stereotypes and user group models.* Information about user groups, either available *a priori* as stereotypes (e.g., Rich [1979; 1983; 1989], Paliouras et al. [1999]) or dynamically calculated as user group models (aka 'communities') (e.g., Orwant [1995], Paliouras et al. [1999]) can be maintained with low redundancy in a user modeling server.
- *Increased security.* Known and proven methods and tools for system security, identification, authentication, access control, and encryption can be applied for protecting user models in user modeling servers (see Chapters 5.4, 6.3.3, Schreck [2003], and Kobsa and Schreck [2003]).
- *Increased support for the holistic design, acquisition, and maintenance of user models*. In the past, many efforts in user modeling research have been devoted to user model representation and inference issues. In commercial settings, however, the main focus is on leveraging the potential of user-related information on an enterprise level, e.g. for improving customer retention rate and brand loyalty [Hagen et al., 1999]. In this vein, areas of work include the
  - i. design of an enterprise-wide user model schema;
  - ii. development and communication of an appropriate privacy policy;
  - iii. acquisition of user-related information at every point of contact with the user throughout the enterprise (e.g., Web site, retail, sales, customer service, direct marketing, call center);
  - iv. integration of complementary user information that is dispersed across the enterprise (e.g., demographic data from client databases, past purchase data from transactional systems, available user segmentations from marketing research, regularities in past purchase behavior found in data mining processes); and finally
  - v. provision of user information to different applications for personalization purposes.

User modeling servers that allow for the (virtual) integration of existing information sources about users and enable access to information stored in user models, can provide the basic platform for such a personalization infrastructure [Truog et al., 1999]. But there is evidence that such an infrastructure can yield advantages in user modeling research environments as well, e.g. for designing and validating methods and techniques in the area of machine learning for user modeling [Pohl and Nick, 1999].

In addition to the aforementioned advantages, many more general ones of centralized systems design (e.g., centralized user modeling servers relieve clients from user modeling tasks and can take advantage of powerful hardware resources), as well as disadvantages (e.g., necessity of a network connection, potential central point of failure), also apply (see for example Goscinski [1991], Tanenbaum [1992], and Orfali et al. [1994]). A discussion must however be omitted here for reasons of brevity.

Despite these potential benefits of centralized user modeling, we believe that current and future usage scenarios for computing devices will require a more sophisticated architecture. These scenarios include

i. *multi-computer usage* (e.g., of a PC at work, a laptop on the go, and a PC at home, whereby the latter two are only temporarily connected to a network),

ii. *mobile computing*, where a user carries a small information device (e.g., a mobile phone, palmtop, or organizer) that can be temporarily connected to a network wherever she goes (access to a computer network, however, cannot always be guaranteed),

iii. *ubiquitous information*, where a user conjures up her information environment at every point of interaction like information walls, information kiosks, and desktops, and

iv. *smart appliances* like intelligent car control systems and household appliances like refrigerators that acquire and manage users' preferences.

These scenarios demand a personalization infrastructure that comprises centralized systems (e.g., user modeling servers), decentralized systems (e.g., 'OPS' profiles[9], user model gatherers [Yimam Seid and Kobsa, 2003], user modeling intermediaries between user modeling servers and adaptive applications), and user modeling systems that are embedded into application systems [Bertram, 2000]. More recent developments in the area of agent-based personalization seem to provide promising concepts and technologies for such a personalization infrastructure. Vassileva et al. [2003] point out that "we can learn from the adaptability, robustness, scalability and reflexivity of social systems to come up with more powerful multi-agent technologies for decentralized applications". We believe that a personalization infrastructure comprises both (virtually) centralized and decentralized components. Regarding the latter, agency may provide a promising basis and, as such, a highly desirable complement to the concepts and technologies that underlie our user modeling server. In order to investigate this further, we recommend future research that starts from (and hopefully ameliorates) the benefits of (virtually) centralized user modeling we introduced at the beginning of this sub-chapter.

1.4 Organization of This Work

Against this background, the aim of this thesis is to (i) analyze the requirements that *user modeling servers*[10] must meet to be acceptable both from a *multi-disciplinary scientific perspective* and from the viewpoint of (commercial) deployment, (ii) design and implement a system that meets these requirements, and (iii) verify its compliance with core performance and scalability requirements under the workload of small and medium-sized real-world environments.

[9] OPS (Open Profiling Standard) is a privacy standard proposed by Netscape, FireFly (which has been recently acquired and reportedly discontinued by Microsoft), and VeriSign that enables users to control the local storage and disclosure of their personal data to Web applications [Reagle and Cranor, 1999].

[10] We define a user modeling server as a centralized software system that provides user modeling functionality to several clients. This enables them to automatically adapt their information and services to different user needs. Application systems that adapt to users automatically at runtime are called 'adaptive', whereas systems that can be tailored manually by the system designer (or possibly the user) by changing certain system parameters are called 'adaptable' [Oppermann, 1994].

In the first part of this thesis, we analyze and discuss requirements for user modeling servers, thereby drawing on experience from a variety of research areas including user modeling, user-adaptive systems, database and transaction management, agent communication, management information systems, and marketing research. We develop two requirements catalogues, one comprising more general server requirements (e.g., multi-user synchronization, transaction management, access control) and the other comprising user modeling requirements (e.g., functionality, data acquisition, extensibility and flexibility, integration of external user-related information, compliance with standards, support for privacy). Based on the latter, we subsequently conduct a review of selected commercial user modeling servers and compare and discuss our findings. Apart from the novelty of such a comparison both inside and outside the classical user modeling literature, the presentation and discussion of the core features of these commercial systems may provide a source of information and inspiration for the design, implementation, and deployment of future user modeling systems in research and commercial environments.

In the second part of our work, we develop an architecture for our user modeling server that complies with the aforementioned requirements catalogues. In order to determine an appropriate server basis, we compare and evaluate common directory and database management systems. Based on the potential benefits directory management systems can provide (they have never been used before as a basis for user modeling servers), we subsequently develop a generic architecture for our user modeling server that consists of a directory server for data management and several ‘pluggable’ user modeling components, each of which implements an important user modeling technique. Finally, we sketch several present and likely future avenues for user modeling and argue that our user modeling server can support these user modeling scenarios as well.

In the third part of this thesis, we prove the validity of our generic server architecture by instantiating a user modeling server for ‘Deep Map’, a project aimed at the development of a portable personalized tourist guide for the city of Heidelberg. We start with a brief presentation of the specific user modeling requirements that we identified in this project. We subsequently describe the user modeling server we developed with a focus on the user modeling components and the learning techniques employed therein. We argue that by integrating these user modeling components in a single server, we can leverage several synergistic effects between, and compensate for well-known deficits of, the learning techniques we adopted from the area of machine learning for user modeling.

In the fourth part of our work, we present the most important results of the experiments that we conducted to empirically verify the compliance of our user modeling server with core performance and scalability requirements introduced earlier. We start with a brief description of our testing approach and the empirically verified real-world workload we simulated in our experiments. We present selected results and discuss potential strengths and weaknesses of our server. As a main result, we argue that our user modeling server complies with the aforementioned criteria in small and medium-sized application environments at moderate costs in terms of hardware resources. In the following chapter, we revisit our requirements catalogs again, thereby arguing that our server provides adequate support for the rather broad range of requirements we collected. Although these requirements are covered to a different degree, we believe that our server clearly excels both the academic and commercial systems we reviewed. In the final chapter, we summarize our main findings and present some lessons learned from deploying our server to real-world environments. Regarding scalability, we describe an experiment we carried out in a high-workload environment. Our results suggest that our user modeling server can be successfully deployed to these environments as well, still at reasonable costs in terms of hardware resources. Finally, we briefly summarize promising avenues for future work.

I
Requirements for User Modeling Servers

In this part of our thesis, we elaborate requirements for user modeling servers. Thereby, we distinguish between

i. *server-related* requirements and

ii. *user modeling-related* requirements.

In the following chapter, we briefly present server-related requirements that we collected from research areas related to user modeling (e.g., database and transaction management, distributed systems). The considerable experience these research areas already acquired regarding design, implementation, and deployment of server technology allows us to restrict our presentation to a rather brief overview. We show the relevance of each requirement for user modeling and apply it to one or more academic user modeling servers.

In the subsequent chapter, we present user modeling-related requirements and apply them to selected commercial user modeling servers. Since there seems to be very little awareness about these commercial servers in user modeling research, we review and discuss these systems in greater detail.

Based on these two requirements catalogues, we design, implement, and evaluate our user modeling server in the remainder of this work.

2 Server-Related Requirements

2.1 Review Methodology

For eliciting *server-related requirements*, we pursued the following threads of investigation:

- Analysis of existing user modeling servers. We screened the literature on several research prototypes (e.g., BGP-MS [Kobsa and Pohl, 1995; Pohl, 1998], Doppelgänger [Orwant, 1995], GroupLens [Konstan et al., 1997], and TAGUS [Paiva and Self, 1995]) and on commercial user modeling servers (e.g., 'Advisor Solutions Suite' [Blaze, 2000], 'Customer Management' [Blue Martini, 2000], 'FrontMind for Marketing' [Manna, 2000a], 'GroupLens' [Net Perceptions, 2000], 'Gustos' [Gustos, 2000], 'Learn Sesame' [Open Sesame, 2000], 'LikeMinds' [Macromedia, 2000], 'Personalization Server' [ATG, 2000], 'RightPoint' [RightPoint, 2000], 'SelectCast' [HNC, 2000], and 'StoryServer' [Vignette, 2000]).
- Collection of requirements from the literature on database, directory, and transaction management (especially Gray [1981], Härder and Reuter [1983], Bernstein et al. [1987], Heuer and Scholl [1991], Gray and Reuter [1993], Orfali et al. [1994], Saake et al. [1997], and Howes et al. [1999]) and, to a less extent, on agent communication languages [Mayfield et al., 1996; Labrou and Finin, 1997; FIPA, 1998a; FIPA, 1998b]. We selected these research areas for their considerable expertise in designing, implementing, and deploying server systems and related interfaces for communication and cooperation.

Whereas the first thread of investigation followed a bottom-up approach (i.e., eliciting features from server instances), the second implemented a top-down approach (i.e., collecting features that are proposed for classes of server systems from the literature). In the following sub-chapter, we briefly introduce those server requirements that we retained after joining and consolidating the findings of the two threads of investigation. We illustrate each requirement against the background of user modeling and apply it to academic user modeling servers.

2.2 Reviews of Server Requirements

2.2.1 Multi-User Synchronization

Multi-user synchronization addresses the synchronization of several users that concurrently operate on individual user models and group models. With 'users', we refer to administrative users, 'real' users, applications, and to components of the user modeling system itself (e.g., a component of the user modeling server that learns group models by applying clustering to individual user models [Orwant, 1995; Paliouras et al., 1999]). Basically, there are two approaches regarding multi-user synchronization reported in the literature: *iterative* and *concurrent* servers (cf. Stevens [1990], Schmidt [1994], and Tanenbaum [1995]).

Iterative servers typically maintain a single FIFO queue (i.e., First In First Out), which stores clients' requests in order of their arrival. The server accommodates incoming requests by fetching an entry from the queue, processing it, and, if necessary, sending results back to clients. An iterative server always executes one request at a time. Requests that are entered into the queue have to wait until they are selected for processing. The resources necessary for processing clients' requests are rarely controlled by iterative servers. In contrast, concurrent servers process several client requests at a time. In order to provide clients a reasonable (and predictable) response time behavior, they maintain several input queues and control the amount of server resources they use for processing client requests.

An iterative server design is appropriate when the amounts of server resources that are necessary for processing clients' requests can (i) be assumed to be very small and (ii) exhibit a rather small variance across clients' requests. These characteristics apply, e.g., to most implementations of the Domain Name System[11]. In deployment scenarios where these characteristics do not apply, a concurrent server design can be regarded much more appropriate. For database management and transaction management systems, a concurrent server design can even be regarded mandatory. The same can be assumed for user modeling servers, since their services can be assumed to not comply with the aforementioned characteristics for such computationally simple services like the translation of domain names of hosts to IP addresses. Especially the strong representational and inferential capabilities of the academic user modeling servers we investigated (e.g., reasoning in first-order predicate logic in BGP-MS) seem to mandate a concurrent server design. However, we did not find any evidence in the user modeling literature about a concurrent design of these academic user modeling servers; hence, we assume that most, if not all, of them are designed as iterative servers. Such a server design, however, can be regarded as highly inappropriate.

[11] The Domain Name System (abbreviated 'DNS') is a server that is used by applications for translating domain names of hosts to IP addresses.

2.2.2 Transaction Management

Transaction management deals with the synchronization (and recovery) of sets of logically grouped user modeling operations. Transactions can be identified in user-adaptive applications (e.g., a single adaptation is triggered by several queries to a user model) and in internal functionality of user modeling servers as well (e.g., activation and deactivation of stereotypes, drawing of inferences, acquisition of assumptions based on user's behavior). The well-known *ACID* properties (i.e., Atomicity, Consistency, Isolation, Durability) of transactions can be interpreted against the background of user modeling as follows (cf. Gray [1981], Härder and Reuter [1983], and Gray and Reuter [1993]):

- ***A***tomicity means that a transaction is an indivisible unit of work: either all or no accesses are processed by the user modeling server.
- ***C***onsistency means that a transaction transforms a user model from one consistent state into another consistent state. If such a state cannot be achieved (e.g., because of integrity constraints being violated), the user model has to be reset to the state before the transaction started.
- ***I***solation means that a transaction (e.g., one that was initiated by an adaptive application) is not affected by other, concurrently executed transactions (e.g., by inference processes within the user modeling system, or by the simultaneous access of other adaptive applications) with respect to shared parts of the user model. Changes initiated by a transaction are not visible to other transactions until this transaction has successfully ended.
- ***D***urability means that once a transaction has been successfully completed, all changes made to the user model are persistent, i.e. these changes survive even system failures.

In database and transaction management systems, a *transaction manager* supervises the progress and state of a transaction. In order to achieve this, it interacts closely with a *scheduler* that controls the relative order in which concurrent operations are executed. The overall aim of the scheduler is to maximize potential concurrency, yet preventing consistency problems that may arise from several transactions running in parallel (see Fink [1999] for related examples). The synchronization of transactions preserves the properties isolation and consistency. Synchronization can be implemented by various locking strategies on data elements and associated protocols (e.g., the two-phase locking protocol [Bernstein et al., 1987; Bell and Grimson, 1992; Gray and Reuter, 1993]). The preservation of the transaction properties atomicity and durability is the main aim of the *recovery manager*. Its tasks include all aspects of transaction commitment and abortion including the rollback to a consistent database state after a system crash.

In order to provide transactions that adhere to the ACID principle, user modeling systems have to incorporate a transaction manager, a scheduler, and a recovery manager. The sophistication of their implementation, however, depends heavily on the intended deployment scenario and the services offered. If the user modeling system is embedded in a single-user application, then various internal functionality (e.g., activation of stereotypes) has to be synchronized with transactions induced by the application. The scheduler and the recovery manager can be implemented in this scenario in a very rudimentary form. Synchronization can be enforced for example via one or more locks (e.g., separate locks for read and write operations) on the whole user model and recovery can be implemented by managing all user model updates in a persistent buffer until a transaction successfully commits. User modeling servers, however, need to implement more sophisticated techniques in order to synchronize various internal functionality with accesses to user model content from several applications. The aforementioned lock granularity may serve as an example. It is hardly desirable that a single transaction locks the whole user model when operating on a relatively small part of the user model, thereby preventing all other transactions using the same model from being executed. More fine-grained locks (e.g., on an interest partition of the user model, on an assumption about a user's interest) increase potential concurrency but raise at the same time the need for a dedicated locking manager. Its main tasks are to synchronize locking by enforcing a specific protocol (e.g., the commonly used two phase locking protocol) and handle problems that may arise from incompatible locking behavior (e.g. deadlocks). For further information on this subject, we refer e.g. to Gray and Reuter [1993].

In the literature on academic user modeling servers (e.g., BGP-MS, Doppelgänger, TAGUS), we found no evidence that any of these systems provides transactional facilities. Hence, we assume that transaction management is a rather new research topic for user modeling servers. In order to motivate future work in this area, we discuss this issue further at the end of this chapter.

2.2.3 Query and Manipulation Language

Query and manipulation language refers to all aspects of external access to a user modeling server including content language and associated protocol issues. The most important requirements regarding a query and manipulation language include the following [Kobsa et al., 1996; Mayfield et al., 1996; Labrou and Finin, 1997; Saake et al., 1997; FIPA, 1998a; FIPA, 1998b]:

- *Application and domain independence* requires the query and manipulation language not being specifically designed for a specific application or domain.
- *Descriptiveness* allows a client of a user modeling server to specify user model operations in a descriptive manner, i.e. without having to rely on details of the user modeling server implementation (e.g., regarding the representation sub-system used).
- *Set orientation* means that user model operations query and manipulate sets of information objects hosted by the user modeling server, as opposed to a navigational access to these information objects. Set orientation is closely related to the aforementioned requirement of descriptiveness.
- *Exhaustiveness* means that the query and manipulation language should provide access to all representational facilities of a user modeling server. A client should not be forced to use additional languages and interfaces provided by the user modeling server for dedicated purposes (e.g., for querying and manipulating schema information).
- *Computational closure* means that a client should be able to assign the result of a user model operation to a persistent variable and subsequently use this information in further requests to the user modeling system.
- *Reaction conformity* requires that syntactically correct user model operations should terminate and return a finite amount of information from the user modeling server.
- *Efficiency* is related to the amount of computing resources necessary for completing a user model operation. From a scalability point of view, the amount of computing

resources should grow moderately (e.g., in a linear or quadratic fashion [Saake et al., 1997]) with the amount of information affected by a user model operation.

- *Scalable set of generic language elements* refers to the query and manipulation language comprising a small number of language primitives (e.g., log in, search, modify) that allow for uniformly accessing and manipulating user model content. Primitives of the query and manipulation language should be grouped into sets, thereby reflecting clients' communication and cooperation needs (e.g., authentication, interrogation, update). Moreover, it should be possible to enhance standard primitives of the query and manipulation language (e.g., by adding new parameters), as well as create custom language primitives that cater to dedicated needs of user modeling clients (e.g., communicating transactional contexts to the user modeling server, providing custom functionality for creating new user models).
- *Adequate set of operators* that includes (i) comparison operators (e.g., $=$, $\neq$, $<$, $\leq$, $>$, $\geq$), (ii) logical operators (e.g., $\neg$, $\wedge$, $\vee$), and (iii) (optional) operators from relational calculi (e.g., $\cup$, $\cap$, $-$, π (i.e. projection), σ (i.e. selection)).
- The *semantics* of the query and manipulation language should be *formally defined*. This is mandatory e.g. for assessing the equivalence of user model operations.
- *Layered architecture* that comprises the levels of communication (e.g., TCP/IP, UPD/IP), messaging (e.g., KQML[12]), and content (e.g., BGDL[13]).
- *Security* requires that the query and manipulation language should provide facilities e.g. for authenticating user modeling servers and clients and for encrypting the flow of information between them.

An assessment of the query and manipulation languages provided by academic user modeling servers (e.g., BGP-MS, Doppelgänger) is quite difficult, mainly due to the lack of related information in the literature[14]. Whereas research prototypes like BGP-MS seem to rate quite well regarding more general requirements like application and domain independence, descriptiveness, formally defined semantics, layered architecture, and security (see for the latter requirement Schreck [2003]), they rate quite poor regarding more practical requirements like exhaustiveness, reaction conformity, and efficiency. There is considerable evidence in the literature that proposes even a paradigmatic clash between requirements like reaction conformity and efficiency on one hand and the strong representational and inferential capabilities employed by most of these research prototypes on the other hand. Saake et al. [1997] points out that query and manipulation languages that adhere to the requirements reaction conformity and efficiency must not be complete. The inferential capabilities of server systems like BGP-MS that are based on reasoning in first-order predicate logic, however, are complete, albeit the logic is only semi-decidable [Pohl,

[12] KQML (Knowledge Query and Manipulation Language) is a high-level communication language which has been proposed by Finin et al. [1993]. Further work on the KQML core including related protocol issues has been carried out by Labrou and Finin [1997].

[13] BGDL (Belief and Goal Description Language) is a high-level language for representing user model contents. BGDL is a part of the query and manipulation language for the BGP-MS user modeling shell system (see Kobsa and Pohl [1995] and Pohl [1998]).

[14] The available literature covers mainly representational and inferential aspects of these systems. The query and manipulation languages provided by these research prototypes are rather briefly described, if at all. The only effort towards a standardized query and manipulation language for user modeling systems seems to be Kobsa et al. [1996].

1998]. The latter means that clients may have to wait for an arbitrary amount of time for their user model operations being processed by these server systems. In order to limit this to a reasonable extent, both BGP-MS and Doppelgänger accept a time limit from their clients.

2.2.4 Persistency

Persistency is the property of parts of a user model to have an arbitrary lifetime other than a transient one. Persistent user model content survives even unpredictable events such as system breakdowns and hard disk crashes. Saake et al. [1997] propose various dimensions for classifying persistency models, including the following:

- *Definition time*: persistency can be statically defined at development time by the user model developer or dynamically assigned at runtime by clients of the user modeling server and the server itself.
- *Granularity*: the persistency property can be assigned to arbitrary parts of a user model, from the level of assumptions in the representation sub-system of the user modeling server up to the level of user models as a whole.
- *Propagation*: persistency as a property may be explicitly assigned (e.g., by a user model developer) or implicitly inherited through structural relationships between assumptions within the user model.

The academic user modeling servers we investigated support persistency only to a very limited extent. It seems that the only research prototype that provides basic support for persistency is BGP-MS. Clients of this user modeling server can request the persistent storage of complete user models on secondary storage. Besides this, no further support for persistency is provided by BGP-MS.

2.2.5 Integrity

Integrity refers to the condition that predefined domain-dependent integrity constraints must be obeyed for a given user model. Heuer and Saake [1995] propose various dimensions for classifying integrity constraints, including the following:

- *Granularity*: integrity constraints can be distinguished regarding their scope of control, e.g. whether they restrict single assumptions in a user model (e.g., the format of a user's e-mail address) or whether they restrict several related assumptions in several models (e.g., a user's name in her individual user model and in models of user groups the user belongs to must be identical).
- *Static versus dynamic constraints*: static integrity constraints control single assumptions (e.g., the presumable domain expertise of a user maintained in her user model being either 'beginner', 'intermediate', or 'expert'), whereas dynamic constraints control, e.g., transitions between assumptions (e.g., the presumable domain expertise of a user maintained in her user model must not change from 'beginner' to 'expert').
- *Checking time*: simple integrity constraints (e.g., the aforementioned restrictions regarding valid assumptions about user's domain expertise) normally require immediate checking after each presumably problematic user model operation. Checking more complex integrity constraints, however, can often be deferred to predetermined points

in time (e.g., before closing a transaction as a sequence of user model operations) or carried out by the user modeling server on a regular basis (e.g., at the end of the day).

- *Reaction*: in case of an integrity constraint being violated by a user model operation, the user modeling server can reject it. An alternative strategy is to 'repair' the condition that violates integrity (e.g., if the user completely removes her user model via an inspection interface, the user modeling server can remove related information about this user, e.g. in models of user groups this user belongs to[15]).
- *Declaration*: another approach for classifying integrity constraints is regarding their integration with the representation sub-system employed by the server. Quite many user modeling servers provide 'model-inherent' facilities for specifying domain constraints (e.g., regarding allowed values for a specific assumption in BGP-MS [Pohl, 1998]). An orthogonal (and very common) approach in database and directory management systems is to explicitly define and maintain integrity constraints in a separate representation, e.g. by using ECA (Event Condition Action) rules like the following (cf. Dayal et al. [1988] and Saake et al. [1997]):

```
rule CheckDomainExpertise
on after User::setDomainExpertise(Expertise: string);
if ((Expertise <> "Beginner") and
   (Expertise <> "Intermediate") and
   (Expertise <> "Expert"));
do abort;
```

  This rule checks assumptions about a user's domain expertise held by the system each time it is modified. If the new value does not comply with one of the values specified in the body of the rule, then the modification is rejected. For more information on ECA rules including more advanced topics like conflict resolution strategies, we refer to Dayal et al. [1988], Bell and Grimson [1992], and Saake et al. [1997].

Plenty of work on integrity issues has already been carried out in the area of (distributed) database and directory management systems. The academic user modeling servers we investigated (e.g. BGP-MS) provide support for preserving integrity mainly via their model-inherent facilities for specifying domain constraints (see the respective facilities of BGP-MS [Pohl, 1998]). Completely missing in these server systems, however, is their support for gracefully handling (quite common) situations that violate integrity (e.g., automatically deleting multiple occurrences of an object in order to preserve referential integrity, deferred checking of integrity constraints).

2.2.6 Access Control

Access control grants or rejects access to a user model, thereby enforcing a predetermined security and privacy policy. Access is granted and/or denied for *objects* (e.g., users, user groups, roles, applications, components of the user modeling server), which carry out specific *operations* (e.g., read, modify, delete) on *subjects* (e.g., models of individual users, models of user groups, single assumptions in these models). The process of granting access

[15] This is also known as 'referential integrity', a well-known integrity constraint in the area of database and directory management systems.

is sometimes also called ‘(positive) authorization’. Over the last two decades, quite a few access control models have been proposed which authorize e.g.

- *explicitly* (e.g., through dedicated access control rules) or *implicitly* (e.g., through specialization from more general access control rules),
- *positively* (i.e., grant access) or *negatively* (i.e., deny access),
- *strongly* (i.e., authorization can not be overridden) or *weakly* (i.e., a general authorization can be overridden by a more specific authorization), and
- by *positive defaults* (i.e., authorization is granted if not explicitly denied) or *negative defaults* (i.e., authorization is denied if not explicitly granted).

For an overview of access control models, we refer e.g. to Castano et al. [1995], Saake et al. [1997], Howes et al. [1999], and, more related to user modeling systems, to Schreck [2003] and Kobsa and Schreck [2003]. The academic user modeling servers we investigated (e.g., Doppelgänger, BGP-MS) provide basic support for access control. Doppelgänger, the older of the two systems, takes advantage of the Kerberos[16] system for authenticating objects and provides authorization based on access control lists. BGP-MS [Schreck, 2003] relies on the more sophisticated SSL (Secure Sockets Layer) technology for authentication, signing, and encryption and takes advantage of a more flexible role-based access control model for authorizing objects. Moreover, BGP-MS provides support for anonymity, pseudonymity, and modification tracking (for more information on this subject, we refer to Chapter 6.3.3 and Schreck [2003]). Facilities for auditing and resource control, however, are largely lacking in both server systems.

2.3 Discussion

When comparing the findings of the two threads of investigation it became apparent that especially transaction management seems to be a rather new challenge for academic user modeling servers. None of the research prototypes we investigated provided transactional facilities and associated *ACID* properties (i.e., Atomicity, Consistency, Isolation, Durability [Gray and Reuter, 1993]). Due to this fact, we briefly present and discuss this presumably new requirement for user modeling servers in the remainder of this sub-chapter. A thorough discussion, however, goes beyond the scope of this thesis and is left as an open field for future research.

The interfaces of the academic user modeling servers we investigated seem to have one feature in common: each access to a user model is treated as a discrete request, i.e. independent from former and subsequent accesses. The sophisticated internal functionality offered by many user modeling systems (e.g., activation and deactivation of stereotypes, drawing of inferences, preservation of model consistency and integrity) is executed autonomously and is isolated from external accesses.

This isolation practice is in sharp contrast to the cohesion between subsequent user model accesses, which can be found in many personalized applications. Examples from the area of

[16] Kerberos is an authentication service that is based on a private key system (sometimes also called ‘shared secret’ system). As opposed to that, SSL (Secure Sockets Layer) is based on a public key system. For more information on this subject, we refer to Smith [1997], Grant [1997], and Diffie and Landau [1998].

user-adaptive hypermedia which motivate the cohesion between subsequent queries to a user model include the following:

- a single adaptation is triggered by several queries about a user's presumed knowledge (e.g., if one of two concepts are presumed to be known by the user then automatically provide a comparison of the two concepts (adapted from Brusilovsky [1996]));
- an adaptation is triggered by several queries about a user's presumed interests several times on a Web page (e.g., if the user is presumed to be interested in historical details then links to historical information for all points of interest are automatically provided [Fink et al., 1998]);
- several adaptations on related Web pages are triggered by several queries about a user's presumed knowledge about concepts (e.g., according to the user's presumed knowledge about the underlying concepts of a curriculum, use a traffic light metaphor[17] on one page and, connected via a NEXT button, suggest an individual curriculum sequencing on a second page [Weber and Specht, 1997]).

The cohesion between subsequent user model accesses stems from the associated consistency context of an adaptation. Following the examples mentioned above, a consistency context can comprise a single adaptation, several adaptations on a single hypermedia page, or several adaptations on a set of related hypermedia pages. An adaptive system needs to communicate this context to a user modeling system in order to be able to adapt in a consistent manner. The user modeling system in turn, is expected to preserve the cohesion between user model accesses by providing a unique 'sphere of control' [Gray and Reuter, 1993]; internal functionality of the user modeling system and accesses from client applications that read and update the same parts of the user model must not be concurrently executed.

If functionality for communicating and preserving consistency contexts is lacking, as it is the case in many user modeling systems today, then the consistency of adaptive applications is jeopardized. Let's exemplify this for the second example presented above. Assume that the user modeling system is queried several times for the user's presumed interest in historical information for points of interest (e.g., churches, museums, public places). The user modeling system, unaware of the coherence between subsequent calls for the user's interest, activates internal functionality (e.g., an inference) that alters this presumed interest in historical information after the first query. Subsequent queries to the user modeling system now return a different assumption for a user's interest in historical information, probably leading to inconsistent adaptations on a single hypermedia page (e.g., a link to historical information is only provided for the first point of interest and not for the other points of interest and vice versa).

A static and restricted form of transaction support that focuses on the ACID properties consistency and isolation can be found in a number of systems including the PAT-InterBook system [Brusilovsky et al., 1997]. These systems offer applications the opportunity to communicate user model accesses that belong to a consistency context in a

[17] Several user-adaptive tutoring systems use the metaphor of traffic signals for recommending learning units, typically red for content that is not recommended for learning, green for content that is recommended for learning, and yellow for content that is recommended for learning, although the user is presumed to not possess all prerequisite knowledge.

package, which, in turn, is handled by these systems in a consistent and isolated manner. Although useful in many adaptation settings, this approach is rather restricted because of the ACID properties atomicity and durability being not provided and static because of the scope of a consistency context being limited to a single package. More complex adaptation contexts (e.g., adaptations on a set of related hypermedia pages) can hardly be covered by such an approach.

So far, we argued that the loss of cohesion between user model accesses can lead to consistency problems in adaptive applications. The consistency of the internal representation maintained by the user modeling system, however, can be jeopardized as well (for more information on this subject, we refer to Fink [1999]). All user modeling systems, regardless of their architecture (e.g., embedded within the application, client/server), that offer internal user modeling functionality and at the same time no means for communicating and preserving consistency contexts cannot guarantee consistency. This is especially the case for user modeling servers which are intended to serve several applications in parallel.

The lack of transactional facilities in today's academic user modeling servers is quite amazing, given the potential consistency problems that may arise from missing transactional facilities [Fink, 1999] and the paramount importance of consistency for the overall usability of applications [Shneiderman, 1987; Nielsen, 1993; ISO, 1998a, 1998b]. The user modeling server we design, implement, and evaluate in the remainder of this thesis caters to these requirements by providing basic transactional facilities (see Chapters 5.3.2 and 10.1).

3 User Modeling Requirements

In this chapter, we present and discuss user modeling requirements as a complement to the aforementioned server-related ones. We illustrate their relevance by applying them to selected commercial user modeling servers. We work out and compare the deployment-supporting features of these systems and point out potential lines of further developments. In general, very little information can be found about commercial user modeling servers (see Appian [2000b] for a notable exception). Developers and vendors are not particularly eager to supply concrete information to academic solicitors and, to the best of our knowledge, most commercial user modeling servers were not even mentioned so far in the user modeling literature (a notable exception seems to be Fink and Kobsa [2000]).

3.1 Review Methodology

We organized our comparison of commercial user modeling servers as follows:

- *Development and validation of a requirements catalog* for guiding and structuring our review. In several validation rounds, we applied the catalog to the product information collected until then, in order to validate its applicability. Collection of pointers to products and companies from sources like

  i. relevant portal Web sites (e.g., Marketing 1to1 [2000], Personalization[18] [2000], and 1to1Web [2000]),

  ii. consulting and research companies (e.g., Appian [2000a; 2000b], Forrester [2000], Jupiter [2000], and Nielsen Media [2000]),

  iii. producers' Web sites (e.g., Autonomy [2000], Bowne [2000], Manna [2000a] and Net Perceptions [2000]), and

  iv. online (business) magazines and newspapers.

  Moreover, we screened the literature on online marketing and on user modeling for work on user-adaptive systems for e-commerce (see Chapter 1.2 for selected references).

- *Selection of a set of representative user modeling servers.* Our initial research revealed more than 50 products that provide personalization functions. Their scope of application, however, is very heterogeneous, ranging from generic Web development and runtime environments to off-the-shelf customer relationship management systems. We therefore first focused on those products that solely offer user modeling functionality and are sold separately (following this, we had to discard, e.g., BroadVision's 'One-To-One' [BroadVision, 2000] and Microsoft's 'Site Server' [Microsoft, 2000d], where the user modeling part is merely a small component in a comprehensive e-commerce application environment). We also required a certain minimum amount of documentation (whose quality even thereafter was poor in many cases). A second selection criterion was based on the similarity of products and the representativeness of a system for a whole similarity group based on its comprehensiveness and reputation.

  Based on the acquisition and inference methods employed by these systems, we decided to form two main groups: systems that implement *collaborative filtering* and systems that adhere to a *rule-based approach*. From those systems that can be associated with collaborative filtering (i.e., GroupLens [Net Perceptions, 2000], Gustos [2000], LikeMinds [Macromedia, 2000], StoryServer [Vignette, 2000]), we decided to elect *GroupLens* as the representative of the whole category. From the category of rule-based systems (i.e., Advisor Solutions Suite [Blaze, 2000], FrontMind for Marketing [Manna, 2000a], Personalization Server [ATG, 2000]), we decided to stay with *Personalization Server* and *FrontMind*, due to their distinctive strengths and weaknesses. From those systems that we could not associate to one of these groups (i.e., Customer Management [Blue Martini, 2000], Learn Sesame [Open Sesame, 2000], RightPoint [2000],

[18] As of the time of writing, it seems that NetPerceptions discontinued their support of this Web site.

SelectCast [HNC, 2000]), we added *Learn Sesame* to our review list, mainly because of its sophisticated features and its status as an early pioneer of personalization.

- *Review and quality control.* After completing the reviews, producers were asked for their assistance in validating our findings (alas with little feedback), and cross-checks were made with other sources of information that were originally not utilized.

Below is the final version of the requirements catalog, which also features selected product data (for more information on some of the requirements, we refer to Kobsa et al. [2001]):

I. *Company* profile, including name, place of business, and a brief history.

II. *Product* profile, including

  - *functionality* offered (i.e., acquisition of user-related information, user modeling, user-related adaptations);
  - *data acquisition* including the input data, namely
    - user data (e.g., demographics, interests and preferences);
    - usage data:
      - observable usage (e.g., selective actions, temporal viewing behavior, ratings, purchases, other confirmative and disconfirmative actions);
      - usage regularities (e.g., frequency, navigation patterns, situation-action-correlations);
      - environmental data (e.g., software platform, hardware platform, locale);
    - acquisition methods (e.g., acquisition rules, statistics, case-based reasoning, decision trees, neural networks, stereotype reasoning, group model reasoning);
  - *representation* methods (e.g., attribute-value pairs, graph-based representations, production rules);
  - *extensibility* and flexibility, especially with respect to complementary acquisition methods;
  - *integration of external user and usage information* and domain knowledge (e.g., from legacy systems and OPS profiles);
  - *privacy* and compliance with existing or forthcoming standards (e.g., technical support for implementing standard privacy policies as defined for instance by the TRUSTe [2000] privacy branding program, compliance with OPS and 'P3P'[19]), and related technical implications (e.g., inspectability of user model contents);
  - *architecture* (e.g., embeddable into applications, single-tier or multi-tier server component);

[19] P3P (Platform for Privacy Preference Project) is a privacy framework that enables Web sites to communicate their privacy practices, and users to exercise their preferences over those practices. Although similar to OPS to some extent, the main difference is in their focus: the focal point of OPS is on the storage, disclosure, and transport of user data, whereas P3P focuses on the mediation of privacy practices, thereby allowing users and Web services to reach an agreement and ensure that the facilitated release of data is in accordance with that agreement. P3P was proposed by the World Wide Web Consortium [W3C, 2000].

- supported *software and hardware* platforms and related APIs (Application Programmer Interface);
- *client base* and publicly accessible Web sites that employ the product online.

III. *Similar products* that are not being described in detail in this survey.

3.2 Reviews of Commercial Server Systems

In the following, we review selected user modeling servers that are available as standalone products. We thereby adhere to the requirements catalog that was presented in the previous sub-chapter. In order to facilitate orientation, the requirements are repeated as run-in headings in italics within each review.

3.2.1 GroupLens

Company. Net Perceptions [2000; Appian, 2000b] has its roots in work on news filtering systems that started in 1992 at the University of Minnesota with the 'GroupLens' project [Resnick et al., 1994; Konstan et al., 1997]. Group Lens provides users with Usenet news in which they are presumably interested. Postings of low quality (e.g., spams, nonsense) or low relevance are not proposed for reading. Automatic filtering of news is based on ratings of Usenet postings that have been provided by like-minded users. Such affinity groups are automatically built by the GroupLens system based on correlations in users' anonymously provided ratings of news articles. This approach has been called collaborative filtering and more recently also 'clique-based filtering' (cf. Alspector et al. [1997] and Kobsa et al. [2001])[20]. In any case, the underlying filtering approach successfully automated the problem of finding like-minded people in the rapidly growing Usenet communities and providing personalized recommendations from their ratings history. For an overview and an evaluation of various filtering algorithms, we refer to Breese et al. [1998] and Herlocker et al. [1999].

Product. After its foundation in 1996, Net Perceptions obtained an exclusive license for the GroupLens technology from the University of Minnesota. After a short technology transfer phase, a first prototype of the GroupLens product was released in November 1996. Since then, the GroupLens product has been developed considerably further and is currently available in version 4.0. Complementary products that take advantage of the GroupLens recommendation engine are available for e-commerce, knowledge management, online advertising, e-mail marketing campaigns, and for supporting call center personnel in providing personalized advice and suggestions to clients. In parallel to these commercial developments, work on related research issues is still carried out at the University of Minnesota. In the remainder of this sub-chapter, we focus on the GroupLens recommendation engine (sometimes also called Net Perceptions recommendation engine).

Functionality, data acquisition. The GroupLens product comprises a recommendation engine and an associated set of APIs (see Figure 3-1). Via these interfaces, applications can send ratings to, and receive predictions from, the GroupLens recommendation engine. Interest predictions are generated by GroupLens from ratings explicitly provided by users

[20] In the remainder of this work we will use the term 'collaborative filtering' rather than the more appropriate term 'clique-based filtering', mainly because it is commonly used in commercial settings.

(e.g., in on-line forms), from implicit ratings derived from navigational data (e.g., products that the online customer viewed or put into the shopping cart), and from transaction history data (e.g., products purchased in the past). Whereas the first two types of user and usage information can be processed by GroupLens at runtime, past purchase data as well as past ratings can only be taken into consideration at startup time.

GroupLens offers three types of recommendations:

i. *Personal recommendations* are based on *action-to-item affinities* between user actions and product attributes (e.g., a system recommends books on science fiction based on users' past queries for books that belong to this category).

ii. *Anonymous recommendations* exploit *item-to-item affinities* between products the user has already expressed an interest in and potentially relevant products (e.g., the system recommends buying recordable CD-ROM disks since the user put a CD-ROM writer into her shopping cart).

iii. *Fast lookup recommendations* build on *user-to-user affinities* between a user and like-minded users (e.g., the system recommends a book on programming languages because users with similar purchase behavior also bought books on this topic in the past).

Personal recommendations are calculated for a given user from her personal history of ratings and do not take ratings of other users into account (e.g., if a user rated science fiction films highly or purchased several SF films in the past, then GroupLens will recommend another SF film). Personal recommendations presuppose that the system has already been able to collect a meaningful amount of explicit and implicit user ratings that are indicators for a particular interest. If this is not the case, anonymous recommendations can be provided. In this case, a sample of already available user ratings is searched for similar users that correlate in their likes and dislikes with the current user. Based on this set of similar users, predictions about the probable ratings of the current user can henceforth be calculated. For those adaptations that require real-time response behavior (e.g., product up-selling or cross-selling recommendations) fast lookup predictions are provided by GroupLens since finding a group of similar users normally requires considerable computing resources in real-world environments with at least ten-thousands of users. Fast lookup predictions do not take users' ratings history into account but rely on predetermined relations between product attributes and categories. For instance, users who put a camera into their shopping cart will subsequently be offered recommendations on appropriate batteries and films.

Representation, integration of external user and usage information. In order to achieve real-time performance, GroupLens maintains all user-related data in cache memory. If this is not possible (e.g., due to shortage in memory), performance decreases significantly. GroupLens' cache memory is initialized from a Ratings Database, where all user ratings are stored. Applications that intend to communicate *a priori* available user data to GroupLens (such as purchase histories) have to convert them into an appropriate format and store them in the Ratings Database. After launching GroupLens, the database is initially sorted and subsequently loaded into memory. Depending on the data volume, this process is reported to take hours or even days. Besides this initial bulk loading facility for user data, no other technical means are provided for integrating legacy data at runtime. Figure 3-1 provides an overview of the GroupLens architecture and depicts the flow of information between the Recommendation Engine, the Ratings Database, and an Application.

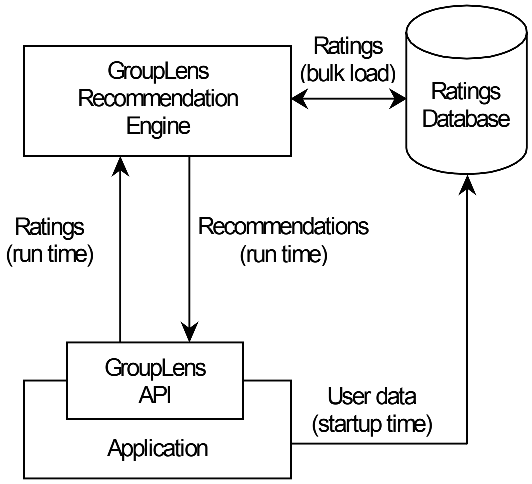


Figure 3-1: GroupLens architecture (based on Net Perceptions [2000])

Extensibility. GroupLens employs various collaborative filtering algorithms for generating predictions. Net Perceptions originally licensed algorithms from the University of Minnesota. During the last few years, Net Perceptions developed this technology further and now owns four pending U.S. patent applications. An example of these developments is a component called 'Dynamic Algorithm Selector', which chooses one out of several competing algorithms for generating predictions based on their performance and accuracy. Competing algorithms operate on the same tasks and run in parallel in the background. In general, performance is traded in for accuracy. Modifications and extensions of the algorithms in GroupLens by third parties are currently not supported and probably also not planned for the future. Besides collaborative filtering, Net Perceptions also claims to employ neural networks, fuzzy logic, genetic algorithms, and rule-based expert systems. However, these methods are not used in GroupLens but rather in the complementary product 'Ad Targeting'.

Privacy. Net Perceptions is actively contributing to and member of several privacy consortia (e.g. TRUSTe). In 1997, GroupLens was one of the first commercial user modeling products that supported the OPS proposal. In principle, GroupLens does not need to know the identity of a person in order to provide recommendations. In practice, however, most of the sites that employ GroupLens require a registration process, since they are not willing to offer the benefits of personalization without a payback in the form of personal data (a notable exception from this is the Web site of CDnow).

Architecture, software and hardware. From an architectural point of view, GroupLens requires a dedicated server with a sufficient amount of memory (i.e., 128 MB or more recommended) and at least one processor that runs on Windows NT or Sun Solaris (IBM AIX support is forthcoming). A distribution of GroupLens across several computers on a computer network is not possible. Connection to a Webserver or e-commerce server is established via an API that is based on CORBA[21]. Local interface support includes Java,

[21] CORBA (Common Object Request Broker Architecture) [Pope, 1997; OMG, 2001] is an industry standard software platform for object communications. At the core of its architecture is the Object Request Broker (ORB), a component that enables software objects to communicate, irrespective of any physical details like location, hardware, operating

C/C++, Perl, COM[22], and CGI[23]. Native database support is offered for Oracle and Microsoft SQL Server. ODBC[24] support that used to be provided by previous versions of GroupLens has been abandoned due to performance reasons.

Client base. Net Perceptions reports more than 190 customers as of March 2000. Web sites that employ GroupLens online include Bertelsmann-BOL, CDnow, E! Online, Kraft, and Ticketmaster Online.

Similar products. There are quite a few commercial systems on the market that can be associated with collaborative filtering. Among the products that are most similar to GroupLens are LikeMinds and Firefly, which has been recently acquired and reportedly discontinued by Microsoft. In addition, there are several companies that hold patents regarding collaborative filtering and associated applications, including Microsoft, IBM, and AT&T. In general, the commercial interest in collaborative filtering seems to be quite high. Appian [2000b] reports that more than twenty U.S. patent applications explicitly discuss collaborative filtering and that nearly 100 describe various kinds of recommender systems. Relevant differences between GroupLens and LikeMinds include

i. its modular architecture, comprising a front-end which generates various predictions based on a user's relation to a set of like-minded users, and a back-end which calculates sets of like-minded users;

ii. its faculty for distributing the aforementioned back-end across a network of computers;

iii. its recommendation engine that is separated into four isolated modules, each working on a different type of input data employed for recommendations (namely purchase data, navigational data, explicitly stated user preferences, and pre-defined product similarities); and

iv. its support for ODBC as a standard interface for database access (native database access is supported for selected database management systems, e.g. Oracle).

3.2.2 Personalization Server

Company, product. Art Technology Group (abbreviated 'ATG' [2000; Appian, 2000b]) was founded in 1991 as a consulting company by computer science graduates from MIT. At the end of 1998, ATG released its product Personalization Server as a complement to their previously released 'Application Server'. Application Server is the indispensable platform for all products from ATG and handles the processing of dynamic Web pages on top of a Web server. Personalization Server extends this functionality by profile management and a

system, and programming language. The CORBA standard is defined and propagated by the OMG (Object Management Group).

[22] COM (Component Object Model) is an object-oriented component software technology that is propagated by Microsoft. COM objects adhere to important object-oriented principles like encapsulation and reusability and can be written in a variety of programming languages including C++, Java, Visual Basic, and COBOL.

[23] CGI (Common Gateway Interface) is a standard programming interface for Web servers. CGI-compliant programs extend the functionality of a Web server and can generate customized Web content dynamically.

[24] ODBC (Open Data Base Connectivity) is an industry-standard application programming interface for accessing relational data sources, including database management systems. In general, applications can access tabular data sources via ODBC, irrespective, for example, of the type of data management system used. ODBC has a wide third-party support due to Microsoft's strong commitment to ODBC on Windows-based software platforms.

rule-based development and runtime personalization environment. Both products together provide the technical basis for complementary products from ATG, e.g. ‘Commerce Server’ for creating online shops and ‘Ad Station’ for providing promotions and advertisements. Figure 3-2 depicts the architecture of ATG’s products suite and outlines some of the products’ interdependencies (Commerce Server and Ad Station are not depicted). In the remainder of this sub-chapter, we mainly focus on Personalization Server, which is currently available in version 4.5.

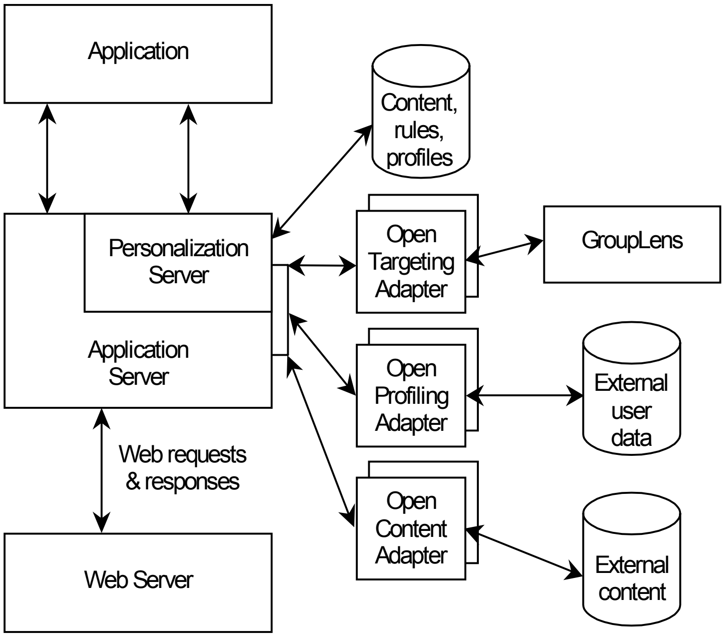
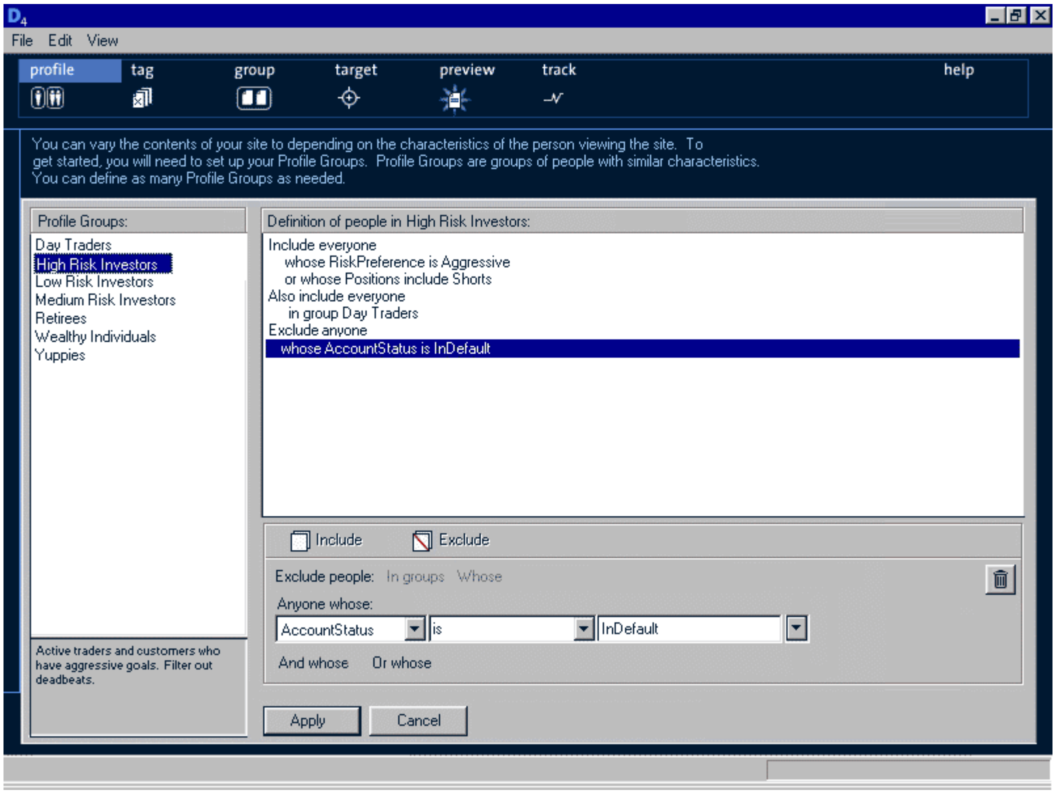


Figure 3-2: ATG architecture (based on ATG [2000])

Functionality, data acquisition, representation. Two main administrative applications are provided by Personalization Server: (i) a development environment called ‘Developer Workbench’, for creating personalized Web pages; and (ii) an administration interface called ‘Personalization Control Center’, to allow non-technicians the definition of personalization rules and the maintenance of profiles. Up to and including the second quarter of 1999, ATG had no U.S. patents concerning their rule-based personalization technology.

Personalization Server’s group profiles comprise relevant characteristics (e.g., age, gender) of user subgroups (e.g., family father, yuppie). Their development and maintenance has to be carried out manually. However, this work has normally not to be started from scratch, since existing user segmentations from marketing and feedback from Personalization Server’s reporting facilities can serve as a basis. Rules that are associated with group profiles allow Personalization Server to assign an individual user to one or more user groups. These rules can take user data (e.g., demographic data like gender and age), implicit information about system usage (e.g., pages visited, products bought) as well as environmental information into account (e.g., domain name, browser type, operating system, available bandwidth). Most of the basic information used within rules is automatically captured by the system (e.g., information about the usage environment).



You can vary the contents of your site to depending on the characteristics of the person viewing the site. To get started, you will need to set up your Profile Groups. Profile Groups are groups of people with similar characteristics. You can define as many Profile Groups as needed.

Profile Groups:

Day Traders
High Risk Investors
Low Risk Investors
Medium Risk Investors
Retirees
Wealthy Individuals
Yuppies

Active traders and customers who have aggressive goals. Filter out deadbeats.

Definition of people in High Risk Investors:

Include everyone
  whose RiskPreference is Aggressive
  or whose Positions include Shorts
Also include everyone
  in group Day Traders
Exclude anyone
  whose AccountStatus is InDefault

Exclude people: In groups Whose

Anyone whose:

AccountStatus | is | InDefault

Figure 3-3: Personalization Control Center [ATG, 2000].
Reprinted with permission.

Figure 3-3 shows the user interface for defining user groups within Personalization Control Center. More specifically, it depicts the definition of a user group called ‘High Risk Investors’: all users who are presumed to have certain characteristics with respect to their ‘Risk Preference’, their ‘Position’, their membership to the user group ‘Day Traders’, and their ‘AccountStatus’, are assigned to this user group.

Group profiles and associated rules resemble very much the stereotype approach that was taken in many research systems (e.g., Rich [1979], Moore and Paris [1992], Kobsa et al. [1994], Ambrosini et al. [1997], Fink et al. [1998], Ardissono et al. [1999]): a group profile forms the stereotype body (which comprises information about users to whom the stereotype typically applies), and one or more associated personalization rules function as activation conditions (or ‘triggers’) for assigning a group profile to an individual user.

Besides group profile activation, rules can also be employed for acquiring user information (e.g., “When a person views Home Equity Loan Information, set DwellingStatus to HomeOwner” [ATG, 2000]), and for providing personalized content (e.g., if a user is interested in football, then provide an article about the prospects of the forthcoming season). The editing and maintenance of rules is carried out via the Personalization Control Center, which puts a graphical user interface with several message windows and drop-down menus for defining dedicated rule parts at the disposal of the developer. Rule formats slightly differ depending on the intended purpose.

Adaptation rules, for example, comprise the following information (for more details including a guided tour of Personalization Control Center, we refer to ATG [2000]):

what; *who*; *when*(optional); *conditions* (optional)

What refers either to an information content category (e.g., football news) or to various selection conditions for information content (e.g., content items whose target audience are football fans). *Who* designates either a user or group profile, or contains one or more selection conditions on profile attributes (e.g., gender is male and income > 60,000 and number-of-visits > 3). The optional component *when* specifies a date for the rule becoming active. An arbitrary number of optional *conditions* forms the last part of a rule (e.g., whether the user's account is with AOL).

Besides this rather simple rule management interface, Personalization Control Center provides no further support for defining and maintaining rules. More sophisticated rule management features are, e.g., offered by 'One-To-One' [BroadVision, 2000] and 'Advisor Solutions Suite' [Blaze, 2000]. Examples of such features include (i) grouping of rules into rule packages (sometimes also called rule sets), (ii) a more fine-grained control over the life span of rules, and (iii) a simulation and testing environment for fielding a rule-based system. These mechanisms are indispensable when, for example, new rules are deployed into a real-world system that is driven by hundreds or thousands of personalization rules. The package mechanism then allows one to organize rules and restrict their scope to, for example, a single package. The life cycle control allows for the control of rules over time (e.g., by specifying an activation date, an expiration date, or a periodic activation). The simulation environment, finally, allows for the testing, logging, and tracing of the personalization system. Apart from this lack of more sophisticated rule management features, another important drawback of Personalization Server seems to be that external user information (e.g., user segmentations from marketing databases, purchase information from transactional legacy systems) cannot be directly referenced in personalization rules.

Extensibility, integration of external user and usage information. Personalization Server offers interfaces called 'Open Targeting Adapters' (see Figure 3-2) for integrating complementary user modeling products and custom extensions into their rule-based user modeling approach (e.g., GroupLens from Net Perceptions). Moreover, Personalization Server provides interfaces called 'Open Content Adapters' and 'Open Profiling Adapters' (see Figure 3-2 for both adapter types) that allow for the integration of user information that is external to ATG's products (e.g., purchase data from legacy systems, user segmentations from marketing databases, demographic user data from customer databases). As mentioned above, incorporating such kinds of external information in personalization rules does however not seem to be envisaged in Personalization Control Center.

Privacy. During our research, we found no commitment of ATG to a dedicated privacy policy (e.g., for handling visitors' session data on their own Web site). We also found no information about ATG's membership in (or active contribution to) privacy consortia, nor about Personalization Server's compliance with established privacy proposals in this area (e.g. OPS). A company white paper [Singh, 2000] advocates the free collection (albeit not release or vending) of user behavior data. When comparing these efforts to those of their competitors (e.g., Net Perceptions, Bowne) it seems that ATG is rather insensitive with respect to privacy and is not very well able to support customers in developing an appropriate privacy policy and implementing appropriate technical means to enforce it.

Architecture. Personalization Server's architecture (which is backed by Application Server) supports server clusters and associated features like

i. *load balancing:* in order to maintain a consistent performance, new user sessions are delegated to available servers and, if none is available, new servers can be dynamically added to a server cluster;

ii. *over-capacity contingency:* in case of traffic peaks, user requests are gradually delayed and finally rejected in order to prevent the whole system from crashing;

iii. *request fail-over:* in case of a server crash, subsequent server requests are redirected to available servers within the cluster;

iv. *session fail-over:* in case of a server crash, a user's session can be transparently migrated to another available server within the cluster without any loss of data; and

v. *geographic fail-over:* if an entire server cluster becomes unavailable, subsequent server requests can be automatically redirected to another server cluster, probably running in a different geographic location.

Software and hardware. The deployment of Personalization Server to real-world settings seems to be supported very well. ATG's products are entirely written in Java and run on top of Windows NT, Sun Solaris, and IBM AIX. For storing content, personalization rules and profile information about users and user groups (see Figure 3-2), all major JDBC[25]-compliant database management systems are supported including those from Oracle, Sybase, Informix, and Microsoft (for a detailed list of supported database management systems, we refer to ATG [2000]).

Client base. ATG lists more than 150 customers on their Web site up to and including the first quarter of 2000. BabyCenter, BMG Direct, living.com, and Newbridge Networks are some of the Web sites that employ Personalization Server online. Other important customers include Sun and Sony.

Similar products. There are several commercial systems that are similar to Personalization Server, most notably One-To-One from BroadVision [2000] and StoryServer from Vignette [2000]. In these systems, however, the user modeling components are embedded and form a rather small part of the overall product functionality (which additionally includes for example transaction handling, content management facilities for products, editorials, advertisements, and discussion groups). Relevant criteria that distinguish Personalization Server include

i. its Open Content and Open Profiling Adapters that allow the integration of external user-related information,

ii. its scalable architecture that is based on server clusters, and

iii. its compliance with industry standards like Java, Java Beans, Enterprise Java Beans, and Java Servlets (see Sun [2000a] for an overview), which seems to support deployment to real-world settings very well.

[25] JDBC (Java Data Base Connectivity) is an industry-standard application programming interface for accessing relational data sources, including database management systems. In this aim, it is quite comparable to ODBC. The most important difference between the two is that JDBC caters solely to the needs of application programs written in Java. JDBC is propagated mainly by Sun.

Compared to Personalization Server, BroadVision's One-To-One offers increased support for user identification, authentication, and encryption through standards like SSL, SHTTP, SET, X.509 digital certificates, and RSA encryption (see Schreck [2003] for an overview and a discussion of these standards with respect to user modeling). Moreover, One-To-One offers developers of user-adaptive applications sophisticated rule management and built-in support for user model inspection[26]. The acquisition methods offered by Vignette's StoryServer excel by complementing rule-based personalization with collaborative filtering. In order to achieve this, Vignette licensed a customized version of GroupLens (see Chapter 3.2.1).

3.2.3 FrontMind

Company, product. Manna [1999a,b; 2000a-d; Appian, 2000b] was founded in 1997. The company is headquartered in Wellesley, Massachusetts, while research and development are located in Tel Aviv, Israel. Manna released its product FrontMind for Marketing (abbreviated 'FrontMind') in the first quarter of 1999, and version 2.0 in April 2000. FrontMind provides a rule-based development, management and simulation environment for personalized information and personalized services on the Web. FrontMind distinguishes itself from other rule-based products like Personalization Server (see Chapter 3.2.2) by having Bayesian networks for modeling users' behavior integrated into their product. Manna owns a pending U.S. patent application concerning this technology. Although currently focused on personalizing Web applications, efforts are underway to deploy FrontMind to call center and online direct marketing scenarios (see Chapter 1.2).

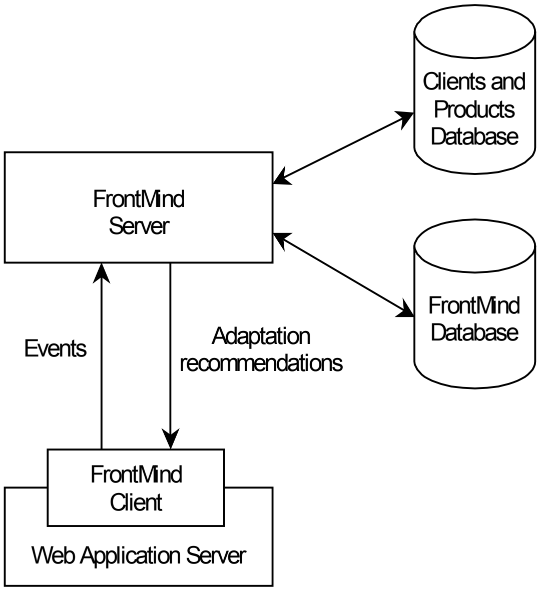


Figure 3-4: FrontMind architecture (based on Manna [2000b])

[26] According to Appian [2000b], however, it seems that many customers of BroadVision do not take advantage of this feature, since they do not want to put their clients in control of their profiles.

Functionality, data acquisition, representation. FrontMind comprises the following components (see Figure 3-4):

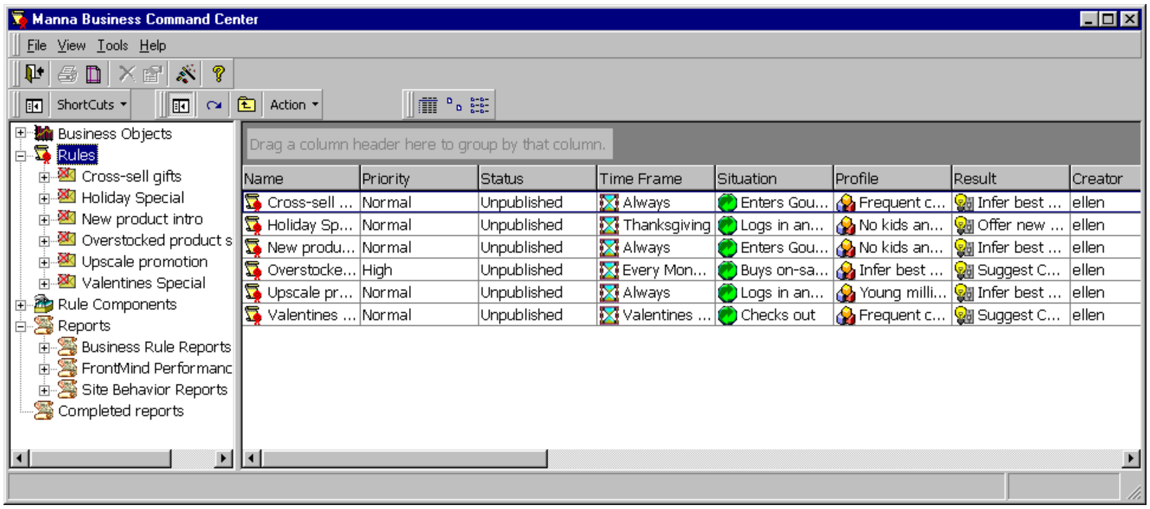
- *FrontMind Client* interfaces with Web application servers and communicates user-related events (e.g., user logins, requests for information on a specific product) to, and receives adaptation recommendations (e.g., product cross-selling recommendations) from, the FrontMind Server.
- *FrontMind Server* provides a rule-based personalization environment that is based on the sub-components *Rule Evaluator* and *Learning and Inference Engine.* The Rule Evaluator receives (a configurable set of) user-related events from the FrontMind Client (see above) and matches them against those adaptation rules that are active at the time of evaluation. Information about users and products from the Clients and Products Database and dynamically acquired information about customers' behavior (see below) can be taken into account during the matching process. The first two sources of information are exploited in a deterministic way, whereas the latter one is used in a 'non-deterministic' (i.e., probability and similarity based) manner. After having selected those rules that match all preconditions, their associated adaptation part is evaluated and the results are communicated back to the FrontMind Client. The Learning and Inference Engine acquires and maintains models of user behavior. Relevant dimensions in these models include referral Web pages, system's usage (e.g., product information requests), purchase histories, and demographics (e.g., gender, age). Internally, these models are based on Bayesian networks. Selected parts of the domain (e.g., purchases of consumer electronics and professional products) can be represented in single models. If appropriate, these models can be arranged in a model hierarchy, thereby creating more comprehensive models (e.g., product purchases regarding appliances in general). Other model topographies (e.g., model chains) can be established as well (for more information on the agent-based background of these behavior models, we refer to Barnea [1999]). According to Manna [1999a; 1999b], designing and implementing behavior models is one of the main tasks that have to be accomplished when deploying FrontMind to customer sites.

  Besides employing behavior information for online adaptation purposes, FrontMind offers a variety of tools for offline model construction, analysis (e.g., unsupervised clustering for customer and market segmentations), simulation (e.g., 'what-if' decision support), and visualization (e.g., dependency graphs).
- 'Business *Command Center*' (not depicted in Figure 3-4): an administration environment for non-technicians that allows for the definition and maintenance of adaptation rules and the simulation of their performance (e.g., matching rate) on the basis of historical data. An additional component reports the performance of business rules (e.g., their matching rate across user profiles, time periods, and product categories) and of the FrontMind system as a whole (e.g., page views, purchase patterns). The log files that support these simulation facilities are stored in the FrontMind Database along with the aforementioned adaptation rules.
- '*Business Object Developer*' (not depicted in Figure 3-4) is a tool that allows technicians the development of business objects, the basic building blocks that constitute every adaptation rule. Standard business objects are provided, e.g. for accessing user profiles, behavior models, and the system's usage history. Internally, business objects can contain (i) database queries, (ii) calls to stored procedures hosted

by database management systems, (iii) calls to the Learning and Inference Engine, and (iv) custom code written in Java for appropriately processing information from the aforementioned sources. Business objects are stored in the FrontMind Database.

- '*Console Manager*' (not depicted in Figure 3-4): administration tool for the FrontMind system that allows for the configuration of the FrontMind Server, the establishment of security policies, and for the tracking of system resource utilization.

The Business Command Center, whose main screen is shown in Figure 3-5, is the central administration interface for controlling personalization in FrontMind. The navigation tree in the left frame comprises the top-level sections Business Objects, Rules, Rule Components, Reports, and Completed Reports. In the remainder of this sub-chapter, we focus on Business Objects and Rules. For more information on FrontMind's reporting facilities, we refer to Manna [2000a].



Name	Priority	Status	Time Frame	Situation	Profile	Result	Creator
Cross-sell ...	Normal	Unpublished	Always	Enters Gou...	Frequent c...	Infer best ...	ellen
Holiday Sp...	Normal	Unpublished	Thanksgiving	Logs in an...	No kids an...	Offer new ...	ellen
New produ...	Normal	Unpublished	Always	Enters Gou...	No kids an...	Infer best ...	ellen
Overstocke...	High	Unpublished	Every Mon...	Buys on-sa...	Infer best ...	Suggest C...	ellen
Upscale pr...	Normal	Unpublished	Always	Logs in an...	Young milli...	Infer best ...	ellen
Valentines ...	Normal	Unpublished	Valentines ...	Checks out	Frequent c...	Suggest C...	ellen

Figure 3-5: Business Command Center [Manna, 2000b].
Reprinted with permission.

The section Business Objects comprises standard objects provided by FrontMind (e.g. 'Customer') and custom objects that may have been defined using the Business Object Developer. The Rules section, which is unfold, contains the adaptation rules that are currently defined in FrontMind. Rules details are shown in the right frame.

Time Frame denotes the times when a rule should become active (e.g., always, at a distinct day, for a period in time, on recurring dates and periods). *Situation* specifies those sets of usage events that trigger an adaptation rule (e.g., user login, information request for a particular product, when a product is put into the shopping cart). An adaptation rule gets activated by the Rule Evaluator only if the Situation matches the actual system's state. *Profile* denotes one or more conditions that refer to attributes in the current user's profile (e.g., whether user's gender is female, the customer's lifetime value is high, the customer already ordered a particular product in the past) or to Business Objects that employ models of users' behavior (e.g., infer those user characteristics that describe potential buyers for a particular product and check whether these characteristics match the current user's profile). *Result* determines the outcome of a rule, i.e. either a set of deterministic adaptations (e.g., show the advertisement for a particular product, propose complementary products for those already in the shopping cart) or non-deterministic adaptations that are (partially) driven by models of users' behavior (e.g., recommend the best-selling product for a particular category). The other rule attributes, *Name*, *Priority*, *Status* and *Creator*, should be self-explanatory.

Several major differences become apparent when comparing FrontMind with Personalization Server regarding data acquisition and representation:

- FrontMind maintains dynamic models of users' behavior, which can take arbitrary user and usage related information into account, whereas Personalization Server relies on rather static group profiles and associated acquisition and activation rules.
- FrontMind employs rules mainly for adaptation purposes[27], whereas Personalization Server also utilizes rules for acquiring assumptions about the user and for assigning profiles of user groups to individual users.
- Besides static user and usage related information, FrontMind's adaptation rules can also take advantage of users' behavior models.

Extensibility, integration of external user and usage information. FrontMind offers no dedicated high-level interfaces for integrating external user-related information (e.g., from a marketing database) and complementary user modeling products (e.g., recommendations from a system that relies on collaborative filtering). On the programming level, however, such external information sources can be quite organically integrated by encapsulating all necessary access details in Business Objects. Once this has been accomplished, these objects can henceforth be transparently employed in rules.

Privacy. Manna's privacy efforts are comparable to those of ATG: hardly any clear information about privacy policies, about compliance with established privacy proposals (e.g., OPS, P3P)[28], and about Manna's membership in or active contribution to privacy consortia. It seems that Manna does not consider privacy being a real challenge for their business and those of their customers;[29] hence, privacy efforts are rather limited and confined to a fairly general privacy statement.

Architecture. FrontMind relies on an agent-based communication and cooperation framework that allows for a very flexible management of software components, including their dynamic distribution across a network of computers [Manna, 2000b]. FrontMind's

[27] In all FrontMind brochures and white papers that we had at our disposal, we found no example of a rule being employed for a purpose other than user-related adaptation, although this should be possible from a technical point of view.

[28] In Manna [2000b], we found the following statement: "The FrondMind Framework ... has a reliable and extremely secure server framework, which supports all major network security protocols, and privacy initiatives such as the Platform for Privacy Preferences (P3P) Project". Without any additional explanation and substantiation, this statement is not very satisfactory.

[29] The only comment on privacy we found was that "the main challenge e-businesses face today in implementing solutions is not addressing customer privacy issues (best handled with a privacy statement to customers affirming what data will be captured and how that data will be used)..." (see Manna [2000a; 2000c]).

exceptional flexibility is supported by a set of architectural features, including the following:

- *plug-ins:* are components of the FrontMind Server (e.g., Business Objects, adaptation rules) that can be created, executed, updated, and removed in real time without interrupting the server's operation;
- *load balancing:* automatically distributes user sessions according to resource requirements across a network of computers;
- *messaging:* all communication between software components is established via explicit messages that are represented in XML[30];
- *events:* the number and type of events that are communicated from the FrontMind Client to the server can be configured by the system administrator. All user actions that are at the disposal of the Web application server can be selected to become input for FrontMind.

Software and hardware. FrontMind's sophisticated architecture and its Java-based infrastructure seem to support its deployment to real-world settings very well. The FrontMind Client can connect to Web application servers via Active X[31], servlets[32], or sockets[33], depending on the operating system platform. The FrontMind Server can run on top of Windows NT and Sun Solaris. The minimum hardware configuration for the server comprises a dual processor board with at least 512 MB RAM and 5 GB of free disk space. On Solaris, the amount of memory should not fall short of 1 GB. With regard to database management systems, FrontMind supports Oracle version 8.0 (or higher) and Microsoft SQL Server 7.0 (or higher).

Client base. Up to and including the first quarter of 2000, Manna reports six important customers on their Web site [Manna, 2000a]: Harcourt General, Saleoutlet.com, Get-Outdoors.com, Entertainment Boulevard, and GourmetMarket.com. There is no information available whether these companies really employ FrontMind online.

Similar products. For a list of products that are similar to FrontMind, we refer to the corresponding paragraph in Chapter 3.2.2. Features that set FrontMind apart include: (i) the integration of dynamic models of users' behavior that are based on Bayesian networks into a rule-based approach, (ii) an agent-based communication and cooperation framework that allows for a flexible management of software components, and (iii) an interface for non-technicians (i.e., Business Command Center) that provides rule management, simulation, and reporting facilities.

[30] XML (Extensible Markup Language) is a standardized markup language for the World Wide Web. Unlike HTML, XML allows for the definition of new markup elements within documents. Defining and applying custom markup elements to objects allows the communication of object semantics in addition to object content. XML is propagated by the World Wide Web Consortium.

[31] Active X is a software technology from Microsoft Corporation that facilitates the exchange of information between software components, applications, and Web pages. Active X builds on another proprietary component technology from Microsoft called COM.

[32] Servlets are CGI-compliant programs that are written in Java.

[33] Sockets are a widely used application programming interface for remote inter-process communication via Internet protocols (i.e., TCP, UDP, IP) and other transport protocols (e.g., OSI TP/IP, X.25, DecNet, AppleTalk).

3.2.4 Learn Sesame

Company, product. ‘Open Sesame’ [Appian, 2000b; Bowne, 2000; Open Sesame, 2000] was a former division of Charles River Analytics, which carried out contract research for various institutions over the last decade in the areas of intelligent agents, neural networks, and expert systems. Their first product ‘Open Sesame!’ was launched in 1993. Open Sesame! was a learning agent for the Macintosh platform that monitored a number of user action types at the interface level (e.g., opening and closing folders, documents, and applications) and proposed the automation of detected repetitive patterns [Caglayan et al., 1997]. Patterns were found along two dimensions: time (e.g., a user’s Web browser is opened every day at 9:00 a.m.) and action sequences (e.g., the user usually launches a dictionary application before opening a text document).

Despite some successes of Open Sesame! (the company claims having shipped more than 35,000 copies), further developments aimed at significantly broadening the scope of deployment beyond a single hardware and software platform. The overall aim was to diversify Open Sesame! towards a user modeling server that could be easily integrated in applications, particularly in Web applications. Besides architectural requirements, this objective challenged especially Open Sesame!’s learning functionality with issues like scalability, robustness, demand for efficient incremental learning, and controllability by client applications. Subsequent developments led to a new generation of learning algorithms, a corresponding U.S. patent application, and to a new product called Learn Sesame that was launched in 1997. During the second quarter of 1998, Open Sesame was acquired by Bowne Inc. The rationale behind this takeover was to strengthen Bowne’s Consulting and Development branch by integrating Open Sesame’s personalization experience. Learn Sesame was available from Bowne both as a standalone product and as a component that was integrated into their Internet applications for the financial sector. In March 2000, Allaire corporation [Allaire, 2000] announced the purchase of Open Sesame from Bowne[34]. In the following, we mainly focus on Learn Sesame (for more details of Open Sesame!, Learn Sesame, and a brief description of the transition from Open Sesame! to Learn Sesame, see Caglayan et al. [1997]).

Functionality, data acquisition, representation. Learn Sesame relies on applications for collecting implicit and explicit user, usage, and environmental data. Relevant usage characteristics (e.g., keywords of requested hypermedia pages, ratings of products, keywords entered in a search form) have to be collected by applications and sent to the user modeling server along with relevant user characteristics (e.g., user id, age, gender, sex, income). Learn Sesame analyzes this stream of time-stamped events for recurrent patterns, and supplies applications with evidences for regularities (e.g., a user’s presumed interest in outdoor clothing, a correlation between the amount of money spent and suitable product categories, a correlation between product category and user demographics like age, gender, income, and formal education for a group of users).

Learn Sesame’s learning algorithms are based on incremental hierarchical clustering[35]. Clusters of events and sequences are identified, analyzed, and so-called ‘facts’ are

[34] The consequences of this acquisition for the product are unclear as yet. As of the time of writing, it seems that Allaire discontinued the development of Learn Sesame.

[35] In contrast, Open Sesame! used neural networks that were based on adaptive resonance theory [Caglayan and Snorrason, 1993; Snorrason and Caglayan, 1994].

generated that describe relevant characteristics of clusters found. Examples include the number of events that ‘support’ (i.e. are in) a cluster, or the relative size of a cluster in comparison to the other clusters as an indicator of confidence. In this learning process, recent events have a higher impact on evidences for regularities than older ones. The process of identifying, analyzing, and communicating clusters can be controlled by applications. A few examples may illustrate this:

- A *timer interval* specifies the amount of time between subsequent clustering stages. Choosing a higher value results in less resource consumption but also in the generation of less timely facts, and vice versa.
- A *learning threshold* specifies the minimum number of events that are required for creating a cluster (see below for an example). Choosing a higher threshold results in the slower generation of more solid facts, whereas a lower threshold results in faster generation of less solid facts.
- A *confidence threshold* specifies the minimum confidence for a cluster to be communicated to applications (see below for an example). Choosing a higher value results in the communication of less facts with a higher confidence, whereas a lower value results in more facts with a lower confidence.

As already mentioned earlier, Learn Sesame provides support for incremental learning of user-related characteristics. Figure 3-6 depicts the six steps that make up every learning stage (the numbers used refer to those in the figure):

1. An application generates events comprising user and usage data, e.g. the following five events about users’ interest in products, each of them comprising a list of four attribute-value pairs:

   UserId=12, ProductId=47, Price=Medium, Trendiness=Low
   UserId=47, ProductId=97, Price=High, Trendiness=High
   UserId=12, ProductId=17, Price=Medium, Trendiness=Low
   UserId=12, ProductId=31, Price=Medium, Trendiness=Low
   UserId=10, ProductId=99, Price=Low, Trendiness=Low

2. Incoming events are buffered by Learn Sesame in an event queue, until the next clustering stage starts (which is controlled by *timer interval*).
3. New clusters of events are identified and previously identified clusters are refined.
4. Clusters identified in the previous step are analyzed, as to whether they comply with the predefined quality requirements (see above), e.g.
   - a *learning threshold* of 2 events, and
   - a *confidence threshold* of 0.5.
5. Facts and related attributes that represent the clusters identified in the previous step are generated and stored in a fact repository. The following fact plus attributes can be generated from the events and the learning parameters:

   UserId=12, Price=Medium, Trendiness=Low (*support*=3, *confidence*=0.6).

   The cluster described by this fact contains 3 events and the relative size of the cluster as an indicator for confidence is 3 out of 5 events, hence 0.6.

6. The application can employ facts learned by Learn Sesame for adaptation purposes.

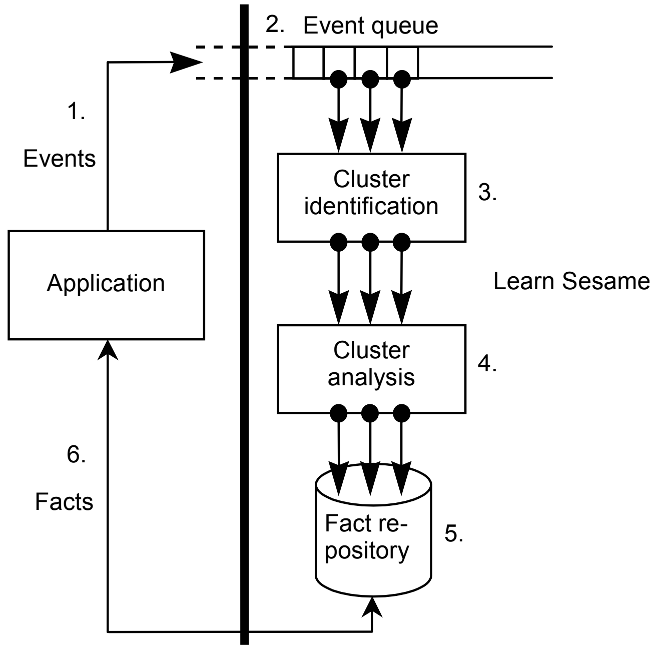


Figure 3-6: Incremental learning process (based on Caglayan et al. [1997])

As can be seen from Figure 3-6, applications and Learn Sesame communicate in an asynchronous manner: events are buffered in the Event Queue for further processing by Learn Sesame and facts are buffered in the Fact Repository for further processing by applications. This allows both applications and Learn Sesame to carry out most of the processing concurrently. In practice, most processes can even be executed in parallel since applications and Learn Sesame normally run on different computers. Likewise, Learn Sesame exploits the inherent concurrency in the cluster identification and cluster analysis processes by delegating concurrent tasks to different program threads whenever possible (see the 'concurrent' arrows on the right side of Figure 3-6). The architecture of Learn Sesame therefore seems to support deployment to real world settings with their inherent demand for performance and scalability in terms of user modeling workload very well[36].

A feature that sets Learn Sesame apart from other commercial user modeling systems is its support for domain modeling by means of a model definition language (MDL). In Learn Sesame, a domain model comprises objects and events. Objects represent domain entities (e.g., the user who is assigned the id number 12, a Web document describing a particular product). Object type definitions represent categories of domain entities and their possible object attributes (e.g., attributes of a Web document like name, creator, modifier, size, date, keywords). Events typically affect one or more objects and are described in terms of changes to objects (e.g., buying a specific product, requesting a Web document describing a particular product). Event types define categories of related events (e.g., buying products, requesting Web documents).

[36] In practice, scalability is often more important than absolute performance since scalability refers to the system's performance relative to an increasing workload (e.g., in terms of number of users, number of events and facts submitted per second, and the size and complexity of the domain model).

The domain model is the indispensable basis for both the applications and Learn Sesame. Applications rely on Learn Sesame's domain model for appropriately assembling and communicating events. Learn Sesame's domain independent learning algorithms rely on the domain model for seizing relevant information about the domain at hand. Especially the meaning of comparability, similarity, and proximity of events is of paramount importance for the clustering process (in the simple example presented earlier, we implicitly assumed a semantics for these concepts). In fact, comparability is defined by the object types of the domain model (i.e., two objects are comparable if they belong to the same object type). Similarity is defined by a subset of object attributes and associated values (i.e., two objects are similar if they are comparable and if they are identical with respect to a subset of attributes). Proximity is defined by the distance between attribute values, measured for example with domain-specific distance metrics (e.g., the time stamps of two Web page requests are regarded as close to one another if they differ by less than a minute, and are assumed to be identical if they differ by less than fifteen seconds)[37]. It seems that despite (or perhaps because) of this expressiveness of MDL, domain modeling has been rigorously restricted in recent versions of Learn Sesame (i.e., versions greater than 1.2) to a fixed model that only comprises a single pre-defined object type (namely 'Web document') and a single event type (namely 'Web document request'). The rationale behind this step was probably to simplify the domain modeling process for Web-based applications.

Extensibility, integration of external user and usage information. Due to the MDL restrictions recently introduced by Bowne, Learn Sesame's learning is confined to the identification and analysis of single attributes only. No further technical means (e.g. APIs) are provided for integrating complementary user modeling products and implementing custom extensions. Integration of events from user and usage information that is external to Learn Sesame (e.g., purchase data from legacy systems, demographic data from a client's database) is supported via a bulk loading interface (see Figure 3-7). As already mentioned, it is however in the responsibility of applications to collect, assemble, and communicate appropriate event vectors from external data. Exporting user and usage data for use with applications other than Learn Sesame (e.g., software for reporting and visualization) is supported via a reporting facility (see Figure 3-7). This facility also provides access to conventional reports.

Privacy. Open Sesame, and now Bowne, committed themselves to a strong privacy policy which is based on the following principles [Open Sesame, 2000; Bowne, 2000]:

- *informed consent*: request permission from a user before acquiring, using, and especially sharing personal data with third parties;
- *anonymity*: a user has the right to remain anonymous (e.g., when visiting a Web site);
- *profile termination*: a user has the right to access, control, and erase any personal information stored about her at any time[38];
- *information security*: all personal information about a user has to be maintained within a secure system.

[37] For more information on Learn Sesame's domain modeling and clustering algorithms, we refer to Caglayan et al. [1997].

[38] For example, Learn Sesame's API offers functionality for deleting selected parts of a user model including the model as a whole.

Bowne is a participant in the P3P project of the World Wide Web Consortium (W3C)[39]. The company claims that Learn Sesame will not only comply to the OPS specification, but to the entire P3P standard after its completion. Bowne encourages its customers to adopt a privacy policy that is similar to their own.

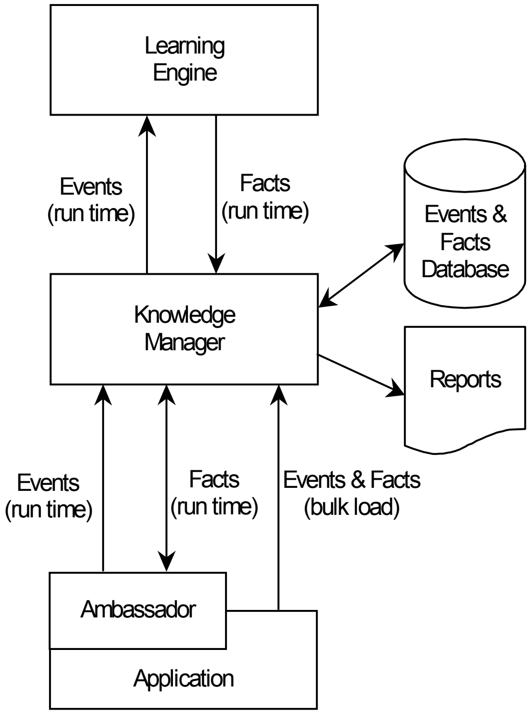


Figure 3-7: Architecture of Learn Sesame (based on Open Sesame [2000])

Architecture, software and hardware. Figure 3-7 depicts the architecture of Learn Sesame, which comprises the following components:

- *Ambassador:* integrates with applications (e.g., Web server, e-commerce server) by offering various APIs for communicating events and facts at runtime;
- *Knowledge manager:* transfers events and facts between the Ambassador and the Learning Engine, implements the Event Queue and the Fact Repository (see Figure 3-6), provides a bulk loading interface for legacy data, and reporting facilities;
- *Learning Engine*: analyzes events for patterns and reports facts describing regularities found.

The communication between these components is carried out via CORBA. The minimum hardware configuration required by Learn Sesame is a single server with a Pentium II processor running at 266 MHz, 64 MB of RAM, and 50 MB of free disk space. Bowne recommends a configuration of at least two servers, each of them being better equipped than the one listed above. Supported operating systems are Windows NT and probably SGI

[39] The World Wide Web Consortium (W3C) develops common Web protocols and aims at ensuring their interoperability. Their freely available specifications called 'Recommendations' have a high impact on the further development of the Web. W3C has more than 400 member organizations from around the world.

Irix and Sun Solaris as well[40]. Connection to a Web server or e-commerce server is established via the Ambassador component, which provides local interfaces for Java, C/C++, and Active X. All ODBC-compliant databases including those from Oracle, Sybase, Informix, and Microsoft can be accessed by Learn Sesame.

Client base. Learn Sesame customers are not listed (nor even mentioned) on the Web sites of Bowne and Open Sesame, with the exception of Ericsson. We assume that the reason for this is a rather small customer base compared, e.g., to the ones of Net Perceptions and ATG. The only Web site that is reported to employ Learn Sesame online is its showcase 'eGenie'.

Similar products. Open Sesame can be considered an early pioneer of personalization, both in research and commercial environments. Despite its early market entry and its sophisticated features, there is, to the best of our knowledge, no commercial system on the market that is comparable to Learn Sesame. In user modeling research, similar work was conducted for example by Orwant [1995] and more recently by Zukerman et al. [1999] in the field of navigation patterns on the World Wide Web. Outside the user modeling community, plenty of related work was carried out in the area of (Web) log file mining. Recent research prototypes are described, e.g., in Fuller and de Graaf [1996], Spiliopoulou and Faulstich [1999], and Cadez et al. [2000]. Examples of commercial products include 'Accrue' [2000] and 'Knowledge Web Miner' from Angoss [2000]. The main difference between these systems and Open Sesame! lies in their focal points: Open Sesame! (and later Learn Sesame) were designed for unsupervised and online learning in (near) real time. Applications can control the nature of patterns learned by Open Sesame! as well as the learning process itself. In contrast, many of the related research prototypes and commercial products have been designed for supervised learning of a relatively small (and often fixed) number of patterns that takes place offline. The implications of this difference are manifold, for example regarding data acquisition and representation (see Chapter 3.3).

3.3 Discussion

In Table 3-1, we summarize the commercial servers reviewed so far along the user modeling requirements introduced in Chapter 3.1 (for reasons of brevity, we omit the requirements 'client base' and 'similar products'). In order to facilitate orientation, we copy the requirement names as run-in headings in italics.

[40] Open Sesame [1998] claims that Learn Sesame supports all three operating systems, whereas Open Sesame [1999] reports that Learn Sesame supports only Windows NT.

Product Requirement	GroupLens	Personalization Server	FrontMind	Learn Sesame
Functionality	User modeling	User modeling, adaptation control	User modeling, adaptation control	User modeling
Data acquisition:				
Input	Predefined usage data, mainly about navigation, ratings, shopping cart operations, and purchases	Predefined user, usage, and environmental data (extensible by custom programming)	Configurable set of usage, user, and environmental data (extensible by system configuration)	Arbitrary usage, user, and environmental data, modeled in MDL[41] and encoded in event vectors
Methods	Collaborative filtering	Simple production rules, stereotypes	Simple production rules, Bayesian networks	Hierarchical clustering (single attributes)
Representation	Implicit, in cache memory	Explicit and relational, in JDBC-compliant databases	Group models: implicit in proprietary files; other user-related data: relational, in dedicated database management systems	Explicit and relational, in ODBC-compliant databases; it is recommended to access this information via APIs only
Extensibility	No	Via Open Targeting Adapters	Via custom programmed Business Objects	No
Integration of external user and usage information	Bulk load at startup time	Via Open Content Adapters and Open Profiling Adapters, dynamically and bidirectional	Anytime, via existing or custom Business Objects; dynamically and bidirectional	Bulk load at any time
Privacy	Company policy, contribution to privacy consortia, OPS compliance	No commitment	Vague policy, declared P3P compliance	Company policy, contribution to privacy consortia, commitment to OPS and P3P
Architecture	Two-tier server	Two-tier server (deployable in a server cluster)	Multi-tier server based on agents that communicate in XML	Three-tier server based on CORBA
Software	Java, C/C++, Perl, COM, and CGI	Java (servlets)	Java (servlets), Active X, and sockets	Java, C/C++, and Active X
Hardware	Windows NT, Sun Solaris, IBM AIX (announced)	Windows NT, Sun Solaris, IBM AIX	Windows NT, Sun Solaris	Windows NT, probably Sun Solaris and SGI Irix

Table 3-1: Summary of reviewed user modeling servers

[41] For more information on MDL (Model Definition Language), we refer to Chapter 3.2.4.

Functionality and input data. For GroupLens and Personalization Server, input data are restricted to predefined information about the system's usage, and, in the case of Personalization Server, also about the user and the usage environment. Custom extensions can be implemented for Personalization Server on top of the monitoring facilities already provided by Application Server. Compared to the restricted set of input data for GroupLens and the rather tight integration of Personalization Server with a single user-adaptive application (environment), FrontMind's configuration facilities for input data and Learn Sesame's domain modeling facilities with their inherent application independence and flexibility seem to be clearly superior. With Learn Sesame, application programmers can communicate information about the domain at hand and control the associated learning process at an appropriate level of abstraction. It is hardly understandable why these powerful domain modeling facilities have been severely restricted. The provision of a transparent MDL wrapper layer that eases domain model development, along with sample configurations for selected deployment scenarios, would have been a more appropriate technical solution to facilitate deployment.

Acquisition methods and representation. With regard to acquisition methods, the reviewed products implement complementary rather than competing approaches. GroupLens uses collaborative filtering, Personalization Server offers (simple) production rules that mainly operate on individual user profiles and stereotypes, FrontMind employs (simple) production rules that take advantage of Bayesian networks, and Learn Sesame employs hierarchical clustering. In the following, we briefly discuss these different acquisition methods along a few key requirements for deployment to real-world settings:

- *Scope of applicability*. Despite the fact that GroupLens supports commonly required user modeling tasks in commercial settings (e.g., predicting user interests and preferences from selected data about system usage), its scope of applicability is rather limited. The acquisition methods used by the other products cover a broader range of user modeling tasks (e.g., Learn Sesame can also learn recurrent patterns of observations in usage data) and can take advantage of a broader range of input data types (namely also data about the user and the usage environment).
- *Facility for incremental learning*. Business practices can often be implemented straightforwardly in rule-driven personalization environments. Moreover, rule-driven personalization allows businesses to be very explicit. From a user's point of view, however, the effects of a solely rule-driven personalization are often found to be quite deterministic [Bachem, 1999; Net Perceptions, 2000]. Unlike non-deterministic recommendations, rule-driven personalization leaves barely any room for users' serendipity[42]. This is mainly due to the fact that the underlying representation system for user information can hardly deal with uncertainty and with changes in user behavior. Keeping track of changing user interests and preferences in real time is, however, a main motivation for user modeling from a marketing point of view (see Chapter 1.2). Even worse, rule design, update and management is primarily a manual process and

[42] With serendipity, we refer to a well-known phenomenon in hypermedia where users, stimulated by unexpected interesting information provided by the system, abandon or sidetrack from the original search goal [Conklin, 1987; Nielsen, 1990]. Similar effects are reported for non-deterministic recommendations provided e.g. by collaborative filtering systems, as opposed to more deterministic recommendations provided e.g. by rule-based systems [Bachem, 1999; Net Perceptions, 2000].

therefore cumbersome and error-prone.
Considering (i) the enormous size of real-world Web sites; (ii) the heterogeneous personalization requirements of a large number of different users and user groups; (iii) the necessity to regularly update both content and personalization behavior; and (iv) the paramount importance of consistency for the overall usability of applications, user modeling servers like Personalization Server that solely rely on rule-based personalization and stereotypes seem to have severe shortcomings. Systems like FrontMind that exhibit both deterministic and non-deterministic personalization behavior seem to have a significant competitive advantage.

- *Explicitly represented knowledge*. In GroupLens, user-related information including dynamically calculated groups of users with similar interests and preferences is represented implicitly in cache memory. Applications can access this information only via pre-defined APIs (see Chapter 3.2.1). Personalization Server and FrontMind maintain their user profiles explicitly in databases. If necessary, applications can access these databases directly, which usually allows for more general exploitation. As opposed to this, FrontMind maintains its models of users' behavior in proprietary files. Likewise, Learn Sesame represents *a priori* available and dynamically acquired user-related information including related evidences (e.g., support, confidence) in proprietary databases. In any case, it is highly recommended to access these proprietary information sources only via the associated APIs.
- *Employing domain knowledge in the learning process*. As already pointed out earlier (see Chapter 3.2.4), Learn Sesame is unique in employing domain knowledge for guiding the learning process, which significantly contributes to the efficiency and scalability of the overall user modeling process.

Extensibility. GroupLens and Learn Sesame do not allow for custom extensions to their acquisition methods, nor do they provide interfaces for user modeling products that offer such kind of functionality. In contrast, Personalization Server allows for such an integration via its pre-defined Open Targeting Adapters (e.g., for GroupLens). Similarly, FrontMind allows for the incorporation of complementary user modeling functionality by custom programming Business Objects. We believe that Manna will make up for this deficiency and offer a dedicated set of Business Objects for integrating complementary user modeling products (e.g., for GroupLens and LikeMinds) in the near future. On ATG's Web site, Accrue [2000] and Macromedia are mentioned as additional technology partners. Both companies offer supplemental products for Web site traffic analysis and reporting (e.g., for investigating the effectiveness of personalization features). We believe that ATG showcases that a focused product which offers open interfaces for complementary tools can successfully cope with the broad scope of requirements for a personalization platform (which range from acquiring user and usage data to user modeling, the provision of user-related adaptations, user model inspection and analysis, and reporting).

Integration of external user and usage information. Leveraging the assets of user-related information that is dispersed across the enterprise (e.g., client profiles, past purchase data, user segmentations from database marketing) seems to be of paramount importance for businesses [Hagen et al., 1999]. The products reviewed support this only to a very limited extent. GroupLens and Open Sesame provide mainly bulk loading facilities for legacy data, whereas Personalization Server and FrontMind can directly integrate user-related information from external sources and vice versa at runtime (in the case of Personalization

Server, this is only of limited value since this information cannot be incorporated in personalization rules). In general, central user modeling servers that allow for the integration of existing information sources about users and, vice versa, enable direct access to information stored in user models, can provide the platform for more holistic personalization from a user's point of view (see Chapter 1.2). In order to achieve this, all relevant user-related information including meta-data has to be (virtually) fused into a single source, regardless of physical details like representation, management system, location, operating system, and network protocol [Truog et al., 1999]. LDAP[43]-compliant Directory Servers and associated meta-directory facilities (e.g., from iPlanet [2000b] and Persistent [2000]) provide such functionality and therefore seem to offer a promising platform for implementing user modeling servers. On top of this platform, dedicated user modeling components can be implemented that communicate with the server platform via CORBA. From a client's and application programmer's point of view, this allows for a transparent access to user-related information and associated user modeling services with commonly available applications (e.g., WWW browsers, e-mail clients) and tools (e.g., directory browsers) via a standardized protocol (e.g. LDAP). For more information on such a user modeling scenario, we refer to Chapter 6.3.1. In Part Two and Three of our thesis work, we design and implement a user modeling server that can integrate various sources of external user and usage information.

Privacy. ATG, and to a lesser extent also Manna, seem to be rather careless regarding privacy, compared for example to the efforts undertaken by Net Perceptions and Bowne. This is somewhat surprising since many tool vendors, their customers and the (online) marketing industry actively propagate and contribute to self-regulation with regard to privacy, in order to prevent governments from passing probably more restrictive privacy laws. The overall aim of self-regulation is to increase customer confidence. Means to this end include (i) privacy initiatives (in the U.S., e.g. from the Federal Trade Commission [FTC, 2000], the Network Advertising Initiative [NAI, 2001], and the Direct Marketing Association [DMA, 2000]), (ii) practices and policies (e.g., Amazon [2000], Bowne [2000], Electronic Privacy Information Center [EPIC, 2000], Net Perceptions [2000], TRUSTe [2000]), and (iii) technologies and standards (e.g., cookies[44] [Kristol and Montulli, 1997], OPS [W3C, 2000], P3P [W3C, 2000]). For an overview on security and privacy issues in user modeling, we refer to Kobsa [2001b], Schreck [2003], and to Chapters 5.4 and 6.3.3.

Architecture. In order to achieve flexibility and scalability, the reviewed user modeling servers take different architectural approaches. GroupLens and Personalization Server both rely on a two-tier architecture (namely a database tier and a user modeling tier), Learn Sesame relies on an additional third tier (for the Knowledge Manager), and FrontMind is based on a multi-tier architecture. Although Learn Sesame takes advantage of an additional third tier, its granularity of distribution and the associated scalability hardly surpasses that of GroupLens and Personalization Server. The main reason is that in all three systems, the 'critical' user modeling functionality is incorporated into a single tier and hence cannot be

[43] LDAP (Lightweight Directory Access Protocol [Howes and Smith, 1997b; Howes et al., 1999; Wilcox, 1999]) is a standardized application programmer interface for accessing information about relevant characteristics of users, devices and services on a network.

[44] Cookies are short pieces of textual information (e.g., a unique ID, a user password) that are used for identifying a computer (i.e., not necessarily a specific user) each time a Web site is visited. Cookies are sent by Web servers and locally stored by browsers.

distributed over a network of computers. Likewise, the employment of differently configured instances of the same user modeling components is currently not supported and, to the best of our knowledge, not planned to be available in the near future. An example that motivates this requirement are different learning strategies that can be accomplished by using differently configured instances of the same learning engine: assumptions about users' interests and preferences can be acquired through rather quick and volatile learning in one Learning Engine, whereas assumptions about users' knowledge that probably require a slower and more solid learning process can be acquired in a second Learning Engine.

A user modeling server thus should allow for (i) the flexible distribution of components across a network of computers, according to resource and availability requirements, (ii) the employment of differently configured instances of a user modeling component, and (iii) the integration of complementary user modeling server products. Due to its sophisticated agent-based architecture, FrontMind should be clearly superior regarding flexibility and scalability (the XML-based communication between agents may not be very efficient though). However, and to the best of our knowledge, FrontMind does not provide replication facilities and Business Objects for integrating complementary user modeling products at the moment. For a user modeling server that caters to these architectural requirements, we refer to Parts Two and Three of this work.

Software and hardware. The APIs supported by the reviewed user modeling servers reflect the targeted kind of user-adaptive applications. Personalization Server provides interface support for Java only (i.e., the programming language used in ATG's products) and, via Application Server, for dedicated Web servers (namely 'Apache HTTPD', Microsoft's 'Internet Information Server', and Netscape's (and iPlanet's) 'Enterprise Server' and 'FastTrack Server'). Group Lens, FrontMind, and Learn Sesame provide a variety of interfaces for several programming languages (e.g., Java, C/C++, Perl). Moreover, they support common component frameworks, e.g. Active X, COM, CGI-compliant Web servers, and one or more e-commerce servers (e.g., BroadVision's One-To-One, Vignette's StoryServer [Vignette, 2000], Microsoft's Site Server, Allaire's 'ColdFusion' [Allaire, 2000], and IBM's 'Net.Commerce' [IBM, 2000a]).

Concluding, we found that the reviewed commercial user modeling servers offer in part sophisticated deployment-supporting features, e.g. regarding the offered acquisition methods and the supported platforms (although the degree of sophistication varies from product to product). Given the potential of centralized user modeling we briefly introduced in Chapter 1.3, we believe that there is considerable room for future improvements of these servers, especially regarding (i) acquisition methods, with a focus on the integration of domain knowledge and method mix at deployment time, (ii) extensibility, and (iii) integration of user-related information that is external to the user modeling server. In the remainder of this thesis, we design, implement, and evaluate a user modeling server that caters to these requirements.

II
User Modeling Server Design

When considering the quite extensive list of requirements we collected so far, it becomes evident that the development of a user modeling server can hardly be carried out from scratch in a single dissertation project. Such an endeavor seems also redundant, given the fact that there are already many server platforms available that cover to some extent the requirements we introduced in Chapters 2 and 3. Following this, we decided to investigate available server systems as to whether they can serve as a basis for our user modeling server. Those requirements that can not be covered by available systems are regarded as necessary extensions. With respect to server systems worthwhile to investigate, we decided to consider (mainly relational) *database management systems* and *directory management systems*.

In the following chapter, we compare and evaluate directories and database management systems (abbreviated databases in the following[45]) against the background of our requirements catalogues. We find that especially LDAP-based directories are generally superior to databases. Based on this finding, we subsequently introduce directory management systems (abbreviated directories), since they are not as commonly known as, e.g., relational database management systems. Subsequently, we design a generic architecture for our user modeling server that comprises an LDAP server for data management and several 'pluggable' user modeling components, each of which implements an important user modeling technique. Finally, we show that this architecture not only caters to present, but also to likely future user modeling scenarios.

4 Server Basis – Directories versus Databases

Directories are specialized database management systems that maintain information about relevant characteristics of users, devices, and services on a network. With historical roots in the sixties and early seventies, two main groups emerged since then (for more information about the history of directories, we refer e.g. to Howes et al. [1999]):

- *Application-specific* directories (e.g., Lotus Notes Address Book [IBM, 2000b], Microsoft Exchange [Microsoft, 2000d]), network operating system directories (e.g., Microsoft Active Directory [Microsoft, 2000c], Novell Directory Services [Novell, 2000]), and special-purpose Internet directories (e.g., Bigfoot [2000], Switchboard [2000]).
- *General-purpose and standards-based* directories (e.g., X.500 DAP[46] [Chadwick, 1996; ITU-T, 2001a], LDAP [Howes and Smith, 1997b; Howes et al., 1999, Loshin, 2000]).

Only the latter group seems relevant for our endeavor, since these systems are not limited to a specific purpose, a particular application, or a specific operating system. General-purpose and standards-based directories are designed to meet the needs of a wide range of applications. They are based on standard protocols that allow for interoperability even between implementations of different vendors and research institutions. Over the last decade, LDAP emerged from this category of directories, because it removed unneeded complexity from X.500 DAP, significantly reduced related resource requirements, and took advantage of the popular TCP/IP, instead of the OSI protocol stack. Nevertheless, LDAP still maintained

[45] We will use the terms directories and databases as short forms for the more appropriate terms directory management systems and database management systems in the remainder of this work.

[46] DAP is an abbreviation for *D*irectory *A*ccess *P*rotocol.

many strengths of X.500, including its information model (see Chapter 5.1), its versatility, and its openness.

Comparing databases with LDAP directories, several differences become evident that recommend an LDAP-based system as a basis for our user modeling server. We briefly review the most important differences in the following sub-chapters.

4.1 Extensibility

LDAP directories provide built-in support for storing and retrieving a variety of user-related information including names, postal addresses, phone numbers, salaries, photographs, videos, digital certificates, passwords, preferences data, and even mobile 'user agents' written in Java (see Kobsa [2001a] for an associated user modeling scenario) in a standardized way. Moreover, support is provided for representing presumably relevant real-world objects like organizations (e.g. 'GMD'), groups (e.g. administrators), and devices (e.g. printers). Directories are not limited to a fixed schema: based on predefined standard types and vendor-specific types of information, arbitrary extensions can be defined in order to cater to specific modeling needs. This includes not only new types of information (e.g., descriptions of user modeling services, user locale), but also custom primitive data types with new semantics (e.g., German telephone numbers, probabilities of users' interest) and behavior (e.g., transient information about user locale that must be periodically refreshed in a user model by a localization application [Yaacovi et al., 1999]).

Comparing LDAP with today's databases, the number of predefined user-related information types and related extension facilities clearly excels those offered by today's database systems. Only a few database systems provide user-related information types and facilities for defining new primitive data types via low-level extensions to the database nucleus (e.g. Informix [2000]). To the best of our knowledge, these facilities are in any case proprietary and therefore hamper interoperability between different database systems and their clients.

4.2 Management of Distributed Information

LDAP directories can manage information that is dispersed across a network of several servers. Historically, this facility catered for deployment scenarios, where administration responsibilities and authorities for directory information are distributed (e.g., the German branch of an international organization is responsible for information about employees in German subsidiaries). To the outside world, however, this distribution is transparent, i.e. the directory appears as a single and consistent source of information. The design and maintenance of large-scale directories provides another important motivation for distributing information. It is often more appropriate to design a large-scale directory as a network of smaller parts (see Figure 4-1). Such a distributed topography often provides a much better performance, scalability, availability, and reliability of the overall service, compared to a single large directory. Moreover, a distributed directory is in many cases cheaper to implement and simpler to manage. For more information on LDAP's distribution facilities, we refer to Howes et al. [1999].

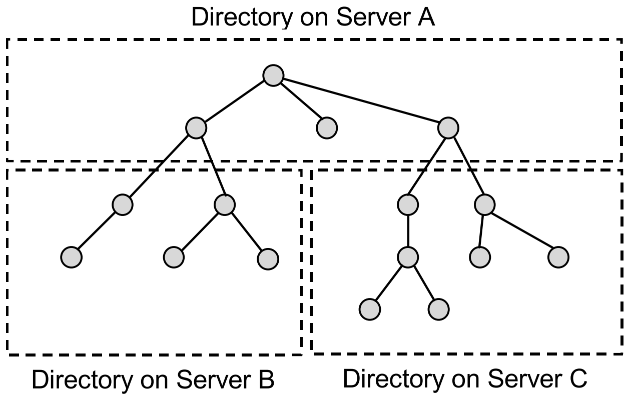


Figure 4-1: Distributed directory (based on Howes et al. [1999])

Database systems can handle data distribution too. The potential granularity and scale of distribution, however, is quite different from LDAP. Databases often restrict the granularity of distribution to the level of database tables and the scale of distribution to a rather small number of sites. LDAP directories are not limited in these respects and support arbitrary levels of granularity and distribution scales.

Against the background of user modeling, LDAP's facilities for managing distributed information seem to be very promising, since they do not only support our aim of developing a user modeling server, but corroborate at the same time promising future user modeling scenarios including the following (cf. Kobsa [2001a]):

- *User models for smart appliances* that maintain relevant user characteristics (e.g., interests and preferences) and adapt their functionality accordingly will come into reality soon. Examples of such appliances include (i) car radios, which store a driver's pre-set stations, volume and tone, and traffic news, (ii) car keys, which adjust the driver seat and the mirrors, and adapt the modality and conciseness of the recommendations provided by their GPS[47]-based navigation system, (iii) mobile phones, which pre-load hypertext pages that are presumably relevant (e.g., stock quotes), (iv) video recorders, which proactively record presumably interesting television programs according to a TV viewer's preferences, and (v) refrigerators, which manage the food stored therein and reorder food that ran out of stock via Internet, thereby taking a user's preferences into account. In order to enable appliances to access an LDAP server, the Connection-less Lightweight Directory Access Protocol (abbreviated 'CLDAP' [Young, 1995]) was proposed as an Internet standard. CLDAP was designed for light-weighted client applications that require access to small amounts of information from an LDAP server. Even more powerful access facilities are at the disposal of those appliances that support a full-fledged LDAP client interface.
  For those appliances that exhibit LDAP server functionalities, LDAP's facilities for managing distributed information can be employed for seamlessly integrating user-related information that is dispersed across several appliances (e.g., preferences

[47] GPS (i.e., Global Positioning System) is a satellite-based system that allows for localizing objects in physical space (e.g., the position of a car, a computer, a hiker).

regarding pre-set stations maintained by a car radio, preferences regarding types of food managed by a refrigerator, preferences for TV channels maintained by a video recorder).

- *Mobile user models* are motivated by recent trends like ubiquitous computing[48], ubiquitous information[49], and agent-based computing, especially in the area of telecommunication and 'intelligent networks' [Magedanz, 1997]. LDAP servers can cater to these scenarios via (i) their remote access protocols (i.e., LDAP and CLDAP), (ii) their facilities for managing distributed information (see above), and (iii) their capability for storing dynamic user modeling functionality (i.e., user agents, either maintained as references to CORBA objects [Ryan et al., 1999a] or as Java objects [Ryan et al., 1999b]).
- *Multiple-purpose user models* have already been extensively discussed in Chapter 3. LDAP servers can fuse user-related information from a variety of sources (e.g., client profiles, purchase information from legacy systems, user segmentations from marketing) in a single (virtual) source, thereby providing the basic platform for a personalization infrastructure (see also Chapter 6.3).

4.3 Replication Scale

LDAP's replication facilities allow for populating and synchronizing a considerable number of copies of directory information (see Figure 4-2).

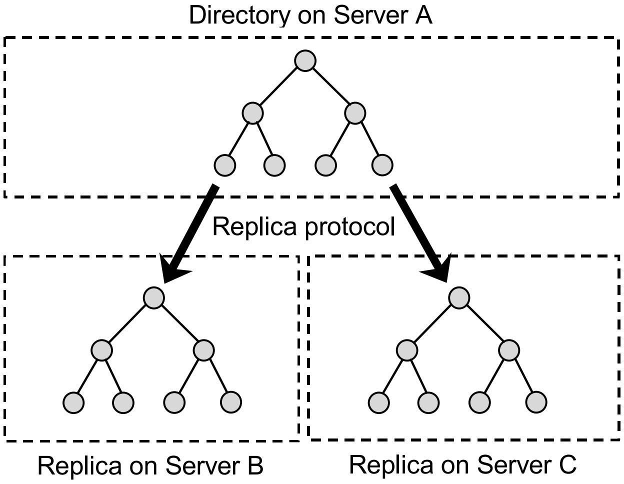


Figure 4-2: Replicated directory (based on Howes et al. [1999])

Historically, replication was mainly motivated by availability and performance considerations when deploying directories to real-world environments. Maintaining replicas

[48] Portable information devices like laptops, palmtops, organizers, next generation mobile phones, etc. provide computing facilities wherever a user goes.

[49] A user can access her personal information environment at every point of interaction (e.g., information kiosk, desktop computer, 'interactive wall').

of directory information can significantly increase the availability and performance of a directory service from a client's point of view. These benefits can also be leveraged when deploying directories to user modeling scenarios:

- *Reliability*. If a source of user-related information is unavailable (e.g., in case of a local user modeling server being down for maintenance reasons), a client can access another user modeling server that maintains a replica of the user-related information maintained by the unavailable server.
- *Availability*. If the network connecting a remote user modeling server and a user-adaptive application becomes (temporarily) unavailable, access to a local replica of user-related information still enables a user-adaptive application to provide personalized information and services. This benefit can be considered to increase e.g. the autonomy of mobile users, smart appliances, and user agents, by reducing their dependence from the availability of a network connection.
- *Locality*. The closer user-related information is to consumer applications, the better the quality of service and, in some cases, the level of security that can be obtained. Creating a local replica of a user model may positively contribute to the security of user-related information, since network communication can be reduced to the amount of communication necessary for keeping replicas synchronized. For a related user modeling scenario, we refer to Chapter 6.3.3.
- *Performance*. Performance may be increased by adding additional replicas to the overall system. E.g., user agents avoid network traffic by taking advantage of a local replica or a user modeling server nearby, instead of accessing a remote user modeling server.

It is worthwhile to note that LDAP's replication and distribution facilities may yield disadvantages as well (e.g., consistency problems in user-adaptive applications that stem from temporary inconsistencies between a user model and its replicas, computing resources needed for resolving replication conflicts). Furthermore, directory replication requires careful planning and deployment. For more information on this topic, we refer to Howes et al. [1999].

Two main differences become apparent when comparing LDAP's replication facilities with those offered by today's databases, namely scale of replication and replication strategy. LDAP directories are replicated on a far greater scale than databases. In a large international organization for example, directory information about employees may have hundreds or thousands of replicas, which are distributed in subsidiaries all over the world. In contrast with this, only a few databases support replication, typically with a few number of replicas. The second important difference is replication strategy. Replication between directories is normally loosely consistent, i.e. temporary inconsistencies between replicas are acceptable. Presupposing a limited amount of computing resources available, applying a weak consistency strategy is the sine qua non for directory replication on a larger scale. Databases in contrast, normally support strong consistency, i.e. database replicas have to be in sync at all times. Maintaining such a strong consistency, however, requires a considerable amount of system resources (we refer also to our discussion of transactional consistency in Chapter 2). This is one of the main reasons why databases normally support only a small number of replicas.

4.4 Performance and Scalability

Directories are designed to cater to the requirements of a wide variety of applications (e.g., e-mail servers and clients, Web server applications and browsers, groupware servers and clients, lightweight database applications). Performance and scalability is of paramount importance for directories, since the number of applications that take advantage of directory information is often not known at deployment time. From a workload point of view, directories are optimized for processing search operations; their performance regarding updates is considered less important. Another factor that differentiates directories from databases is their lack of sophisticated transactional facilities. Only a few directories provide simple support for transactional consistency, the scope of synchronization, however, does rarely exceed a single LDAP operation (for more information on this topic, we refer to Chapter 5.3).

Databases on the other hand, are designed for a relatively small set of dedicated applications. Performance and scalability is important for database systems too. Their presumable workload, however, differs considerably from the one for directories. Databases are designed for a workload with rather balanced ratios for search and update operations (as this is the case in many merchandise transactions). Regarding search performance, Shukla and Deshpande [2000] report that databases excel directories with (i) an increasing number of entries that match a certain query and (ii) an increasing result set for a certain query. If the number of matching entries and the overall result set is small, however, directories have shown in their evaluation a far better search performance than databases. Apart from performance considerations, databases are assumed to offer sophisticated facilities for safeguarding transactional consistency (see also our discussion of these facilities in Chapter 2).

Against the background of our endeavor of designing and implementing a user modeling server, it seems of paramount importance to decouple server design as much as possible from the one of its applications. And reflecting the rather tight coupling between databases and their client applications, directories seem to be provide a more open basis for our user modeling server than databases do. To the best of our knowledge, there is no evidence in the current user modeling literature that motivates (i) a high update frequency for user data (e.g., demographics, skills and capabilities, knowledge, interests and preferences), (ii) the need for retrieving large amounts of user information, and (iii) more sophisticated transactional facilities (see also Fink [1999]). A quite different situation may be encountered when processing information about a systems usage (e.g., temporal viewing behavior, navigation patterns, ratings, purchases, situation-action-correlations, software platform, hardware platform, locale). Usage data, however, are normally not persistently stored, but (more or less) immediately processed by associated user modeling functionality. See Kobsa et al. [2001] for an overview of related systems and Chapter 8.4 for a user modeling component that processes certain types of usage data.

4.5 Standards

The last important area where directories significantly differ from databases is standards. The most important areas of standardization for LDAP include the following:

- *LDAP protocol specifications* for version 2 and 3, i.e. RFCs[50] 1777-1779 [Yeong et al., 1995; Howes et al., 1995; Kille, 1995] and 2251-2256 [Wahl et al., 1997a; Wahl et al., 1997b; Wahl et al., 1997c; Howes, 1997; Howes and Smith, 1997a; Wahl, 1997]. Some of these standards are, in turn, based on X.500 standards, e.g. RFC 2256, which defines the syntax and matching rules to be used for attribute types and object classes in a LDAP user schema is based on X.501 [ITU-T, 2001b], X.520 [ITU-T, 2001c], and X.521 [ITU-T, 2001d].
- *Proposed extensions to LDAP version 3* include RFC 2589 [Yaacovi et al., 1999] for managing dynamic LDAP entries that need to be periodically refreshed by client applications in order to persist, RFC 2820 [Stokes et al., 2000] for common requirements towards interoperable LDAP access control models, RFC 2713 [Ryan et al., 1999a] for LDAP schema elements that represent Java objects, and RFC 2714 [Ryan et al., 1999b] for schema elements that host CORBA object references.
- *Related Internet standards or proposed standards* that have been adopted by LDAP include RFC 2222 [Myers, 1997], which specifies SASL (Simple Authentication and Security Layer). SASL is a generic framework for negotiating security parameters (e.g., for authentication, encryption, and signing) between applications. LDAP version 3 provides native support for SASL.
- Besides the aforementioned SASL, LDAP employs additional *security standards* such as X.509 certificates (as defined in RFC 2559 [Boeyen et al., 1999a] and RFC 2587 [Boeyen et al., 1999b]) and the SSL (i.e., Secure Sockets Layer) and TLS (Transport Layer Security) protocol. The latter is defined in RFC 2246 [Dierks and Allen, 1999].
- A common, text-based *format for representing and exchanging directory content* called LDIF (i.e., LDAP Data Interchange Format) has been proposed recently for standardization. LDIF is specified in RFC 2849 [Good, 2000].
- An LDAP *programming interface* for C has been proposed in RFC 1823 [Howes and Smith, 1995], an API for Java is available as an Internet Draft [Weltman et al., 2001]. Moreover, there are a number of proprietary and (mostly) freely available SDKs (Software Development Kit) for a variety of languages including C, C++, Java, Perl, and Basic. Examples include the Directory SDKs from Netscape [2000b], JNDI (Java Naming and Directory Service) from Sun [2000b], and ADSI (Active Directory Services Interface) from Microsoft [2000c].

Although there are a few standards for databases (e.g., for SQL [ISO, 1989; 1999]), their number and scope falls far short of those for directories. The implications of this lack of standardization are manifold, the most important seeming that no real interoperability can be achieved between database systems of different vendors (e.g., an Oracle client application can be presumed to not work with a Sybase database).

[50] RFCs (Request for Comments) describe many aspects of Internet-based computer communication, e.g., networking protocols, procedures, programs, and architectural concepts [IETF, 2000]. RFCs may also contain meeting notes (e.g., of the IETF) and opinions regarding the aforementioned topics.

Adherence to open standards, however, can be considered crucial for user modeling clients and servers, since this fosters their interoperability. On a more technical level, this is mainly achieved by decoupling

- the design and development of user-adaptive applications from the one for user modeling servers,
- particular design and development decisions that have been taken during the development process of clients and servers (e.g., regarding security of user-related information), and by decoupling
- the design process for user modeling servers and clients from decisions about a particular implementation (e.g., clients are not affected by the server being migrated to a server product of a different vendor).

Until recently, only a few standardization efforts have been undertaken in the user modeling community, mainly regarding the application programming interface to user modeling systems. Kobsa et al. [1996] propose an interface that is based on KQML, and Kummerfeld and Kay [1997] discuss common interfaces including LDAP for user modeling purposes. Interestingly, Kummerfeld and Kay ruled out LDAP, mainly because of its lack of client-side schema retrieval and manipulation features. Meanwhile, these features have found their way into LDAP version 3 (see RFC 2252 [Wahl et al., 1997b]).

Summarizing the aforesaid, directories seem to excel databases regarding extensibility, management of distributed information, replication scale, performance and scalability, and adherence to standards. User modeling servers and clients that take advantage of directory technology can be assumed to exhibit a considerable degree of openness and flexibility. In Chapter 6.3, we further substantiate this claim by showcasing the deployment of our directory-based user modeling server to several present and (potential) future user modeling scenarios.

5 Introduction to LDAP Directories

In the previous chapter, we argued that LDAP-based directories provide a promising basis for our user modeling server. Therefore, we present LDAP now in a more stringent manner and focus especially on those features we did not cover so far. In the following, we introduce LDAP by means of the following four models:

- *Information model*, which defines the types of data that can be stored in a directory.
- *Naming model*, which describes how to organize and refer to directory data.
- *Functional model*, which prescribes how to access directory data.
- *Security model*, which defines how to control access to directory data.

For more information on LDAP, we refer to Howes and Smith [1997b], Howes et al. [1999], and to the RFCs mentioned in Chapter 4.5.

5.1 Information Model

The basic unit of information in an LDAP directory is an *entry*. An entry represents information about a (real-world) object (e.g., users, servers, printers, organizational units). Associated with an entry are *attribute*-value pairs, which describe relevant object properties (e.g., name, address, description). An example is the user model entry for the hypothetical user Peter Smith, which looks in LDIF format as follows (for a more thorough discussion of a slightly different version of this entry, we refer to Chapter 8.2.1):

```
dn: cn=Peter Smith, cn=User Model, ou=UMS, o=gmd.de
objectclass: top
objectclass: person
cn: Peter Smith
age: 36
sex: m
continent: eu
description: department manager
```

Each attribute (e.g. `description`) is associated with an attribute type, which defines (i) a name and an object identifier (abbreviated 'OID'), (ii) an indicator whether one or more values are allowed for an attribute, (iii) an attribute syntax, (iv) a set of matching rules, (v) an indicator whether an attribute is intended to be used by the system or the user (i.e., category *operational* or *user*), and (vi) possible restrictions on the range or size of attribute values.

For the aforementioned attribute `description` for example, the attribute type definition looks as follows (in ASN.1 notation [ITU-T, 1997]):

```
description ATTRIBUTE ::= {
WITH SYNTAX
DirectoryString {1024}
EQUALITY MATCHING RULE caseIgnoreMatch
SUBSTRINGS MATCHING RULE caseIgnoreSubstringsMatch
ID 2.5.4.13 }
```

This type definition

- restricts values for `description` to a single text string with a maximum length of 1,024 characters (as specified by the ASN.1-based syntax description `DirectoryString {1024}`),
- associates the matching rules `caseIgnoreMatch` for equality and `caseIgnoreSubstringsMatch` for substring matching to attribute values (both ASN.1-based matching rules specify that case of letters and leading, trailing, and multiple spaces are ignored), and
- assigns `2.5.4.13` as an OID to this type definition (actually, this OID has been assigned to this standard attribute type by the X.500 standards committee).

Each directory entry belongs to one or more *object classes* (e.g. `person`). Each object class defines (i) a class type, (ii) a set of mandatory attribute types, (iii) a set of optional attribute types, and (iv) an object identifier. For example, the object class `person` already mentioned earlier is defined as follows (in ASN.1 notation):

```
person OBJECT-CLASS ::= {
SUBCLASS OF {top}
MUST CONTAIN {sn | cn}
MAY CONTAIN {description | seeAlso | telephoneNumber |
             userPassword}
ID 2.5.6.6}
```

This object class definition

- specifies that `person` is a subclass of `top`,
- defines `sn` (i.e., surname) and `cn` (i.e., common name) as mandatory attribute types,
- determines `description`, `seeAlso`, `telephoneNumber`, and `userPassword` as optional attribute types, and
- assigns `2.5.6.6` as an OID to this class definition (`person` is a standard object class defined by the X.500 standards committee).

If an entry belongs to more than one object class (e.g., printer and server), then the set of applicable mandatory and optional attributes is simply calculated as the union of the two object class definitions. Potential inheritance conflicts are avoided by (i) prioritizing mandatory over optional attributes, (ii) not providing overriding, and (iii) maintaining a flat namespace (i.e., inheritance is only structural). The same principles apply also to inheritance between object classes in LDAP.

5.2 Naming Model

The LDAP naming model mandates the organization of LDAP entries representing users, organizations, services, etc. in an inverted tree structure. In this respect, LDAP's naming model resembles a hierarchical file system (e.g., in UNIX), where each directory comprises files and sub-directories. Besides this similarity, however, there are also a few differences between LDAP's naming model and a hierarchical file system:

- The root entry of an LDAP tree is a special entry that contains server-specific information (e.g., about LDAP versions supported, operations provided, entries hosted, security features implemented, and alternative servers that can be contacted in case of a breakdown). Besides this information, no application-specific data can be placed in the root entry of an LDAP tree. In a hierarchical file system, however, the root directory is the common ancestor and comprises all files and directories contained therein.
- A second difference is that entries in an LDAP tree contain data (represented in attribute-value pairs) and can have at the same time child entries underneath them. In a hierarchical file system, however, each node is either a file or a directory, but not both. Only files may contain data and only directories may contain sub-directories. Compared to this, directory entries exhibit file and directory characteristics at the same time.

- The final difference between LDAP and a hierarchical file system is regarding naming of individual nodes within the tree. LDAP naming is in little-endian order (i.e., the least significant component is written first), whereas naming in a hierarchical file system is in big-endian order (i.e., the most significant component is written first). In other words LDAP naming is backward relative to file system naming. An example in LDAP can be given as follows:

  ```
  cn=Peter Smith, cn=User Model, ou=UMS, o=gmd.de
  ```

  This name, also called ‘*distinguished name*’ or abbreviated ‘*DN*’, refers to the user model of Peter Smith. The whole name is constructed by appending the names of the parent entries of Peter Smith and separate them by commas, back to the root entry. The leftmost name component of an entry is called ‘*relative distinguished name*’ or abbreviated ‘*RDN*’ (i.e., `cn=Peter Smith` in our example). For unambiguously referring to an entry, all RDNs that share the same immediate parent entry have to be unique (e.g., no second entry named `cn=Peter Smith` is allowed below `cn=User Model`).

LDAP supports a hierarchical organization of directory entries. No further restrictions, e.g. regarding specific hierarchy topographies or object classes for entries, apply[51]. Non-hierarchical topographies can be represented by employing so-called ‘alias entries’. Applying the analogy to the UNIX file system again, aliases are comparable to symbolic links. In LDAP, aliases point to other entries that can be located in the same or a remote directory tree (see Figure 5-1 for an example).

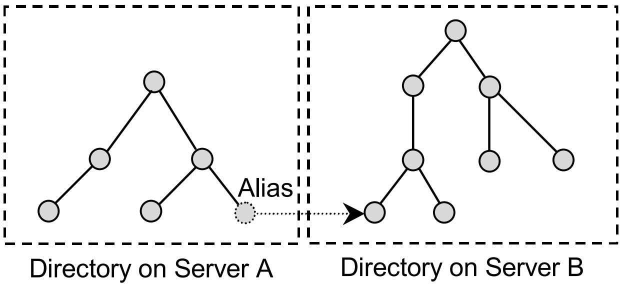


Figure 5-1: Alias connecting two directory trees (based on Howes et al. [1999])

The part of the directory tree that is hosted by a server is called ‘*partition*’ (or ‘naming context’ in X.500). Taking a search operation on the partition hosted by Server A as an example, Server A has to contact Server B as soon as the alias entry has to be checked whether it matches the given search criteria. Additional queries to external servers, however, can be assumed to be quite expensive in terms of resources; hence, not all directory servers support aliases.

Although aliases can be used for connecting partitions that reside on different LDAP servers, LDAP’s facility of choice for ‘intra-linking’ a distributed directory are ‘*referrals*’.

[51] This is quite different from LDAP’s ancestor X.500 [Chadwick, 1996; ITU-T, 2001a], where the topography of a directory is much more mandated (e.g., only entries representing countries, localities, and organizations can be placed immediately below the root entry).

Quite comparable to aliases, referrals are explicit references that connect the different partitions of a distributed directory (for an example of a distributed directory, we refer to Figure 4-1). The main advantage of referrals is that they are standardized as a part of the LDAP v3 specification [Wahl et al., 1997a].

In a distributed directory, each partition is hosted by a dedicated directory server. If a client queries an arbitrary server that hosts a partition of a distributed directory (e.g., Server A in Figure 4-1), the server passes back to the client the result along with referrals that point to additional relevant information hosted by other directory servers (e.g., Server B and C). Subsequently, it is up to the client whether to stay with the returned result or to chase the referrals by re-submitting the query to the additional servers. An alternative approach is that the directory server automatically chases referrals on behalf of the client. In this case, the server forwards the query to potentially relevant servers, collects the results, and passes them back to the client. This approach is called '*chaining*'.

Referring and chaining have both advantages and disadvantages. Clients that take advantage of referrals benefit from a fine-grained control regarding their distributed directory operations and can keep their users informed about the progress of their directory operations, although at the cost of increased client complexity. Chaining significantly relieves clients from the complexity of accessing a distributed directory. From a client's perspective, the whole directory appears as a single and homogeneous source of information. On the server side, however, the workload can be assumed to be significantly higher, since servers have to accomplish additional tasks like querying remote servers, collecting results, and sending them back to their clients.

To the best of our knowledge, there are no servers available that support both referring and chaining. Most of today's LDAP servers support referring, whereas many X.500 servers support chaining. Another approach is to implement the handling of referrals at the client side within the application programmer interface (e.g., using the Netscape Directory SDK [Netscape, 2000b]), thereby relieving clients as well as servers.

5.3 Functional Model

The LDAP functional model comprises operations for accessing a directory. These operations constitute three groups:

- *query operations* allow for searching and retrieving information,
- *update operations* allow for adding, deleting, renaming, and modifying entries,
- *authentication and control operations* allow for authenticating clients and servers and for controlling previously initiated LDAP operations.

Besides these predefined operations, custom ones can be defined by taking advantage of the '*extended operation*' facility. This framework allows for extending the LDAP protocol in a standardized manner [Wahl et al., 1997a].

5.3.1 Query Operations

For searching the directory and retrieving data, LDAP provides the operations *search* and *compare*. LDAP search allows for searching a directory and retrieving matching entries. It takes the following parameters (regarding parameter names, we adhere to RFC 2251 [Wahl et al., 1997a]):

- *BaseObject* specifies the entry where the search should start. The base object is described with its DN, e.g. `cn=User Model, ou=UMS, o=gmd.de`.
- *Scope* specifies the search space, starting from the base object. Three types of scopes can be specified: *base*, *one level*, and *subtree*. Figure 5-2 depicts these scopes and the resulting search spaces for a particular directory tree.

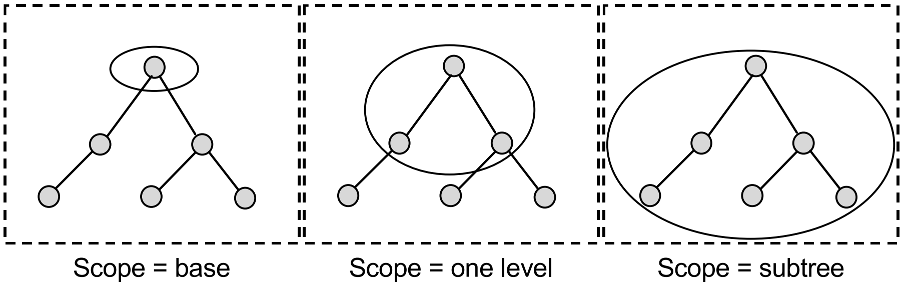


Figure 5-2: LDAP search scopes (based on Shukla and Deshpande [2000])

A scope of *base* limits the search operation to the base object (e.g., search for `cn=Peter Smith, cn=User Model, ou=UMS, o=gmd.de` and retrieve the value of the attribute sex for this person). A scope of *one level* limits the search to the base object including those entries immediately beneath it (e.g., start searching at `cn=User Model, ou=UMS, o=gmd.de` and retrieve the value of the surname attribute of all entries of type person immediately beneath this entry). A scope of *subtree* limits the search to the base object and the entire sub-tree beneath it (e.g., start searching at `cn=User Model, ou=UMS, o=gmd.de` and retrieve the value of the surname attribute of those entries of type person that show a significant interest in football).

- *DerefAliases* controls the dereferencing of aliases during the search operation (e.g., whether the alias depicted in Figure 5-1 is chased during a search operation or not). The LDAP server can be instructed to either chase aliases for the base object, the entries beneath it, or both.
- *Sizelimit* specifies the maximum number of matching entries that should be returned by the server. A size limit of zero indicates that the client requests for all matching entries. Server-side limits, however, may apply and further restrict the maximum number of matching entries returned by the server.
- *Timelimit* defines the maximum time a client can afford to wait for a search operation to complete. A time limit of zero indicates that a client imposes no limit. As already mentioned for size limit, server-side limits may apply and further restrict the time limit. Both limits, size limit and time limit, can be employed by administrators for preventing

the directory against threats like (i) excessively resource-consuming operations, (ii) software bugs, and (iii) denial-of-service attacks.

- *TypesOnly* determines whether the search result should contain either attribute types or attribute names and values. This feature is useful, e.g., when a client wants to check which attributes are actually available for a specific entry.
- *Filter* can contain one or more filter terms. Each term comprises an attribute, an operator, and an attribute value. The term `(sn=smith*)` for example, denotes those directory entries that contain an attribute surname starting with the string 'smith'. The following table provides an overview of standard LDAP filter operator types along with some examples (for a more thorough presentation of LDAP search filters, we refer to RFC 2254 [Howes, 1997]):

Filter Operator Type	Example	Comment
Presence	`(sn=*)`	Matches any entry that has at least one value in its attribute surname.
Equality	`(sn=Smith)`	Matches those entries that have exactly this value in their attribute surname.
Substring	`(sn=*mit*)`	Matches those entries that have the string 'mit' somewhere in their attribute surname.
Approximate	`(sn~=Smithh)`	Matches those entries that have an attribute surname that sounds like the value given. The matching algorithm that implements this operator depends on the server vendor and the languages supported by an LDAP server.
Comparison	`(sn>Smith)`	Matches those entries that have a value in their attribute surname which is lexicographically greater than the value given.

Table 5-1: LDAP search filter operator types

An additional filter operator not contained in Table 5-1 is the extensible match filter. Quite comparable to the extended operation facility discussed earlier, this operator provides a standardized framework for defining custom filters (for more information on this subject, we refer to RFC 2251 [Wahl et al., 1997a]).

In order to constitute more complex LDAP search filters, filter terms can be combined with Boolean operators, i.e. AND (abbreviated ‘&’), OR (abbreviated ‘|’), and NOT (abbreviated ‘!’). The following example illustrates this:

```
(&(objectclass=person)
  (organizationalrole=client)
  (!(sn=Smi*)))
```

This search filter matches those entries of type objectclass person whose organizational role is client and who have an attribute surname that does not start with the string ‘Smi’.

- *Attributes* specifies the list of attributes that should be returned for matching entries. If no attributes are specified or if this parameter contains the wildcard character ‘*’ then all attributes are returned by the server.

The second of the two query operations is LDAP compare. It allows a client to check whether a directory entry contains a particular attribute value. LDAP compare takes the following parameters:

- *Entry* specifies the DN of the entry to be checked (e.g., `cn=Peter Smith, cn=User Model, ou=UMS, o=gmd.de`).
- *Ava* (i.e., *a*ttribute-*v*alue *a*ssertion [Yeong et al., 1995]) contains an attribute name and a value the entry is to be compared with (e.g., `sn` and `Smith`).

If the entry contains the specified attribute-value assertion, then the server returns an affirmative response to the client. Other possible cases are (i) the specified attribute does not exist or (ii) the attribute exists, but does not contain the specified value. In any case, the server returns a respective indication to the client. From a programmer’s point of view, this ability is the main motivation for preferring the compare operation over the search operation.

5.3.2 Update Operations

LDAP provides four operations for modifying directory contents: *add*, *delete*, *rename* (or modify DN), and *modify*. It is worthwhile to note that update operations are atomic, i.e. either all modifications of a single operation or, if not possible (e.g., in case of a potential directory schema violation), no modification are performed by the server. For a review of the conditions that have to be met in order for a specific update operation to succeed, we refer to RFC 2251 [Wahl et al., 1997a] and Howes et al. [1999].

The add operation allows for creating new directory entries. It has two parameters:

- *Entry* specifies the DN of the entry to be added. Aliases are not dereferenced when locating the new entry to be added.
- *Attributes* contains a list of attributes and associated values that constitute the new entry. Mandatory attributes must be included in this list (e.g., those determining the RDN and the objectclass of an entry), operational attributes (e.g., createTimestamp, creatorsName) must not be included, since they are exclusively maintained by the LDAP server.

The delete operation removes leaf entries from a directory tree. It takes a single parameter, the DN of the entry to be deleted. Aliases are not dereferenced when locating the entry to be deleted.

The rename (or modify DN) operation can be used for renaming an entry (i.e., changing its RDN) or for moving a whole sub-tree within the directory topography. It has four parameters:

- *Entry* specifies the DN of the entry to be renamed (e.g., `cn=Peter Smith, cn=User Model, ou=UMS, o=gmd.de`).
- *Newrdn* contains the new RDN for this entry (e.g., `cn=Peter J. Smith`).
- *Deleteoldrdn* controls whether the old RDN is to be deleted or to be retained within a multiple value attribute.
- *NewSuperior* specifies the DN of the new parent entry. If the parent entry remains the same, this parameter can be left blank.

The modify operation can be used for updating attributes of a directory entry. It has the following parameters:

- *Entry* specifies the DN of the entry to be modified. Aliases are not dereferenced when locating the entry to be modified.
- *Attributes* specifies for each attribute the requested operation (i.e., add, delete, rename), the respective attribute name and associated values.

As already mentioned earlier, an update is performed by the server as an atomic operation. Although single attribute modifications may temporarily violate the directory schema, the modified entry has to comply with the schema after the update operation has been executed. If this is not the case, all modifications are automatically rolled back by the LDAP server.

5.3.3 Authentication and Control Operations

LDAP offers the operations *bind* and *unbind* for authenticating clients and servers. The *abandon* operation is endowed for controlling previously initiated LDAP operations.

The bind operation allows for exchanging authentication information (e.g., passwords, certificates) between a client and a server. It takes the following parameters:

- *Version* specifies the LDAP version to be used during a session (e.g., version 3).
- *Name* contains the DN of the directory entry the client wishes to bind as (e.g., `cn=Peter Smith, cn=User Model, ou=UMS, o=gmd.de`). An anonymous bind can be accomplished by providing no DN.
- *Authentication* contains a client's credentials, e.g. a password in case of a simple authentication or a request for Kerberos authentication in case of a SASL-based authentication. As already mentioned earlier, SASL is a generic and extensible framework for negotiating authentication, encryption, and signing services.

The server subsequently verifies a client's credentials for the given DN and, if its identity is approved, grants certain access privileges to the client. These privileges persist until the end of a session or until the client re-authenticates again.

The client can terminate a session at any time using the unbind operation. Upon receipt of an unbind, the server terminates all outstanding LDAP operations, discards any authentication information related to the client, and closes the connection.

The abandon operation can be used by a client for terminating an ongoing LDAP operation (e.g., a long-running search). The abandon operation takes a single parameter *MessageID*, which identifies the operation that should be abandoned by the server.

5.4 Security Model

In this sub-chapter, we start with a brief discussion of potential threats to directory security and user privacy. For a more thorough discussion of security and privacy issues in user modeling, we refer to Schreck [2003] and Kobsa and Schreck [2003]. Besides this, there is plenty of literature, Internet standards, and commercial products that cover various aspects of (directory) security (e.g., Myers [1997], Smith [1997], Diffie and Landau [1998], Boeyen et al. [1999a; 1999b], Dierks and Allen [1999], Howes et al. [1999], Netegrity [2001]).

The purpose of the LDAP security model is to complement general security and privacy guidelines, policies, (best) practices, and laws in protecting directory information against threats like the following (see Schreck [2003] for an overview and Chapter 3.3 for additional references):

- *Unauthorized access*, e.g. through
  - forged or stolen credentials (e.g., passwords, private keys),
  - connection hijacking, where a hijacker (program) pro-actively responds to server requests addressed to authorized clients, thereby preventing them from replying,
  - network sniffing, where an attacker eavesdrops on the information exchanged between an authorized client and a server,
  - 'Trojan horses', which imitate legitimate client programs and, once activated, perform actions compromising security and privacy, e.g. by recording a user's password and sending it to an illegitimate third party,
  - backdoor access, where an attacker uses non-standard access mechanisms to storage systems (e.g., by browsing and searching the database underlying a directory) or gains access to import/export sources of directory data (e.g., customer databases, LDIF files),
  - physical access to the directory server or parts of it (e.g., console, hard disk, directory backups on archive media), and
  - software bugs (e.g., in the directory server, operating system, network management system) that allow attackers to bypass or tunnel security mechanisms (e.g., booting the server from floppy disk or CD-ROM, thereby bypassing standard authentication procedures of the operating system).

- *Unauthorized tampering*, e.g. through
  - a 'man in the middle', i.e. an attacker (program) who interjects itself between a client and a directory server, and
  - various forms of masquerading, where attackers (or attacker programs) imitate directory clients and servers as well.
- *Non-accessibility*, e.g. through
  - illegitimate consumption of directory services, e.g. by a client continuously executing resource-consuming search operations or inserting vast amounts of information into the directory, and
  - illegitimate consumption of hardware resources, e.g. by an attacker absorbing inordinate amounts of CPU time, network bandwidth, or disk space.

A variety of technologies are available for safeguarding against most of the aforementioned security and privacy threats. For a better overview, we associate the more prevailing ones (e.g., SSL (TLS[52]), Kerberos) to one or more of the following categories:

- *Authentication* allows a party to verify another's identity. LDAP's security model offers a standardized interface (see the previous chapter) to various authentication schemes including (i) anonymous authentication (i.e., no authentication), (ii) simple passwords, either communicated as plain text or encrypted via an SSL-secured connection, (iii) X.509 certificate authentication via SSL, and (iv) SASL-based authentication and encryption using e.g. Kerberos. It is important to note that there is an inherent trade-off between the level of security achieved by a certain authentication technology and the computational and organizational efforts necessary for establishing it. Using X.509-based authentication, e.g., requires managing a public key infrastructure. Moreover, there may be additional implications regarding deployability of a directory-based solution. Certain directory clients, e.g., may get disclosed from a directory service, because they do not support a specific security technology. For more information on this topic, we refer to Howes et al. [1999] and Wilcox [1999].
- *Signing* ensures the authenticity and integrity of information exchanged between clients and servers. LDAP's security model supports signing, e.g. through SSL. Within an SSL connection, each block of information is accompanied by a cryptographic checksum that allows clients and servers to (i) verify the sender and (ii) check whether the data has been tampered with while in transit.
- *Encryption* allows for encoding all information exchanged between parties. During the negotiation phase of an SSL connection, e.g., a client and a server agree on an algorithm (e.g., RC4, DES, IDEA) that is to be used for encrypting the flow of information. Besides SSL's encryption facilities, LDAP's security model provides support for alternative encryption services (e.g. MD-5) via its SASL interface. For more information on this topic, we refer to Wilcox [1999].

[52] As of the time of writing, there is very little difference between TLS and SSL; hence, we use the older and more widespread term SSL as a generic term for both technologies in the remainder of this work.

- *Access control* grants anonymous and authenticated clients access to directory information. Besides the requirements for an access control model defined in RFC 2820 [Stokes et al., 2000], there is currently *no standard access control mechanism* for LDAP[53]. In our implementation, we decided to take advantage of the access control model offered by iPlanet Directory Server [iPlanet, 2000b][54], since we employ this commercial server as a basis for our user modeling server (see Chapter 6.2 for further information). iPlanet Directory Server establishes access control via a set of access control lists (abbreviated 'ACL's in the following). Each ACL implements an access control rule and is usually attached to a directory entry via the special attribute `aci` (i.e., Access Control Information). An ACI grants access to the directory entry it is attached to and to all entries beneath it (i.e., its children). Its granularity can be very fine-grained; if necessary, down to a single attribute of a single entry. In general, an ACI comprises the following information (for a more formal presentation of this subject, we refer to the online documentation of iPlanet Directory Server [iPlanet, 2000b]):

  - *directory resources* (i.e. *objects*) the ACI applies to (e.g., entries in a particular sub-tree, entries that match a given search filter, specific entries, specific attributes),
  - *access rights* granted (e.g., read, write, search, compare, add, delete), and
  - *directory clients* (i.e. *subjects*) the ACI applies to (e.g., specific users, anonymous users, non-anonymous (i.e. known) users, all members of a user group, users specified in attributes of object entries, users from a specific IP address or a specific domain, users accessing the directory during a period in time).

  By default, all users are denied access to the directory of any kind. Starting from this, ACIs implement access control rules that grant or deny access. The following ACI, which is presented in LDIF format, allows directory administrators to access and modify the attributes of all directory entries that are below the entry `o=gmd.de`:

  ```
  aci: (target="ldap:///o=gmd.de")
   (targetattr="*")
   (version 3.0; acl "allow everything for administrators";
   allow(all)
   groupdn="ldap:///cn=Directory Administrators,ou=Groups,o=gmd.de";)
  ```

  Due to the paramount importance of access control for directory security and user privacy, we discuss the access control model of iPlanet Directory Server further at the end of this sub-chapter.

[53] Although the access control models offered by today's LDAP servers share some commonalities, there are still many differences that hamper interoperability (e.g., when replicating and migrating access control information from one LDAP server product to another).

[54] During the first quarter of 2002, iPlanet was acquired by Sun. iPlanet Directory Server is now marketed as Sun ONE Directory Server [Sun, 2002a].

- *Auditing* allows for keeping track of all communication between clients and directory servers. Auditing is indispensable, since it allows for determining whether directory security and user privacy has been compromised and in what manner. Comparable to access control, there is currently no auditing standard within LDAP's security model (e.g., for maintaining and searching various log files, monitoring server activity). For the purpose of our work, we rely on the auditing facilities provided by iPlanet Directory Server (for more information, we refer to the online documentation [iPlanet, 2000b]).
- *Resource control* is an appropriate means for preventing the directory server from denial-of-service attacks (e.g., by consecutively executed resource-consuming search operations) and trawling attempts (e.g., by unauthorized bulk downloads of directory data). Like for access control and auditing, we rely for our user modeling server on the controlling facilities of iPlanet Directory Server [iPlanet, 2000b]. A directory administrator can impose various constraints on the amount of resources that are dedicated for processing clients' requests (e.g., the maximum number of entries to be searched, the maximum number of entries returned for a single search, the maximum number of seconds a single search operation can take).

In conclusion, LDAP's security model provides *standardized support for authentication, signing, and encryption*, while providing *no standard facilities for access control, auditing, and resource control*. For the latter group, we rely on the respective facilities offered by iPlanet Directory Server.

We conclude this sub-chapter with a few examples for access control, thereby (i) 'gluing' together the four LDAP models we introduced so far and (ii) showcasing some access control scenarios our user modeling server can support. The first feature we would like to further explore is using search filters in ACIs (for more information on search filters, we refer to Chapter 5.3.1). In the following example, read and search access to users' (presumably public) coordinates (i.e., a set of specific user attributes as shown below) is granted to anyone, i.e. anonymous users and authenticated users, depending on an user's permission. The latter condition is controlled by the hypothetical attribute `coordinatesAccess` in users' profiles. This attribute can be managed by each user, probably starting from a default value set by an administrator. The following two ACIs implement this policy:

```
aci: (target=”ldap:///cn=User Model,ou=UMS,o=gmd.de”)
  (targetfilter=”(coordinatesAccess=public)”)
  (targetattr=”sn||cn||postalAddress||telephoneNumber||mail||jpegPhoto”)
  (version 3.0; acl “publish users’ coordinates”;
 allow(read, search, compare) groupdn=”ldap:///anyone”;)
aci: (target=”ldap:///cn=User Model,ou=UMS,o=gmd.de”)
  (targetfilter=”(coordinatesAccess=private)”)
  (targetattr=”sn||cn||postalAddress||telephoneNumber||mail||jpegPhoto”)
  (version 3.0; acl “keep users’ coordinates private”;
 allow(none) groupdn=”ldap:///anyone”;)
```

The first ACI grants access (i.e., read, search, and compare) to a user's coordinates to anyone, if the condition `coordinatesAccess=public` is true. The second ACI explicitly denies access, if the condition `coordinatesAccess=private` is true. If neither of these conditions is satisfied, access to these attributes is controlled by additional

ACIs or, if not present, access is denied by default. The targeted attributes in these ACIs (e.g., mail, jpegPhoto) are partially defined by the object class inetOrgPerson, a proprietary class defined by iPlanet [Smith, 2000]. InetOrgPerson takes in turn advantage of the respective object classes organizationalPerson and Person, as defined in X.521 [ITU-T, 2001d] and in LDAP version 3 [Wahl, 1997]. The following table summarizes the relation between the aforementioned object classes and their respective attributes (for more information on this topic, we refer to Chapter 5.1):

Object class	Inherits from…	Attributes required	Attributes allowed
Person	Top	sn[55],cn[56]	telephoneNumber
OrganizationalPerson	Person		postalAddress
InetOrgPerson	organizationalPerson		mail, jpegPhoto
UmsPerson	inetOrgPerson		coordinatesAccess

Table 5-2: Example of object class inheritance

The last row in Table 5-2 shows the object class umsPerson, which we defined for our user modeling server. umsPerson subclasses several standard object classes, thereby inheriting a set of required and allowed attributes. This preserves compatibility with existing directory-enabled applications as much as possible, while still catering for new attributes like coordinatesAccess.

Using search filters in ACIs offers a variety of advantages. First, it allows users to control access to their profile information without having to explicitly (i.e., using LDAP tools like 'Globus Browser' [Gawor, 1999]) or implicitly (using mediation applications like 'SiteMinder' [Netegrity, 2001]) specify ACIs. Second, it considerably reduces the number of ACIs that have to be inserted and maintained in a directory, thereby considerably relieving administrators from administrative burden.

Search filter-based ACIs can be employed in a variety of access control scenarios including the following:

- Access is granted to entries that belong to certain object classes (e.g., objectclass=person, objectclass=group, objectclass=interests).
- Access is conceded to entries that are associated with a companies' organizational structure (e.g., ou=Development[57], ou=Sales).

[55] sn (i.e. surname) is a standard attribute that contains the family name of a person (see RFC 2256 [Wahl, 1997]).

[56] cn (i.e., common name) is a standard attribute that holds the name of an object. If the object is a person, this attribute contains her full name (see RFC 2256 [Wahl, 1997]).

[57] ou (i.e., organizational unit) is a standard attribute that denotes the name of an organizational unit (see RFC 2256 [Wahl, 1997]).

- Access is granted to entries according to an established workflow model (e.g., `organizationalRole=HelpDesk-of-the-Day`).

So far, we presented scenarios, where search filters determine the set of entries and/or attributes an access control rule applies to. Additional flexibility regarding the subjects an access rule applies to can be achieved by relying on attributes in object entries. If we assume, e.g., the need for establishing proxies for all users in our directory, who can access and update specific profile information on behalf of other persons, we can implement this policy with the following ACI:

```
aci: (target=″ldap:///cn=User Model,ou=UMS,o=gmd.de″)
 (targetattr=″*″)(version 3.0; acl ‟allow proxy access″;
 allow(read, search, compare, write)
 userdnattr=″proxy″;)
```

The subjects to whom this rule applies are determined by the `proxy` attribute of `cn=User Model,ou=UMS,o=gmd.de` and its children. Thomas Cook, for example, is granted access to the profile of Peter Smith, if (i) he is successfully authenticated and (ii) if Peter Smith's profile designates him as a proxy as follows:

```
dn: cn=Peter Smith,cn=User Model,ou=UMS,o=gmd.de
objectclass: top
objectclass: person
cn: Peter Smith
proxy: cn=Thomas Cook,cn=User Model,ou=UMS,o=gmd.de
coordinatesAccess: private
```

In our example, the single ACI shown above is equivalent to a set of ACIs that explicitly denote proxies. The size of this set can be assumed proportional to the number of proxies in an organization (i.e., a potentially huge set). Following this, the main advantage of access control based on attribute values is a significantly reduced number of ACIs and, related to this, considerably less administrative efforts for establishing and maintaining an access policy like the aforementioned one. A smaller number of ACIs implies also less errors, more flexibility in case of reorganization, and better performance of the directory server. Moreover, access control that is based on attribute values can be regarded mandatory in deployment scenarios where users control access to their personal information (e.g., by maintaining the mentioned proxy information in their profile).

When a subject accesses a directory entry, the directory server checks the ACIs attached to this entry and subsequently those ACIs attached to all its parents in the directory tree. As we have seen earlier, ACIs can grant as well as deny access (see our first scenario above, where the second ACI denies access to users' coordinates). In case of a conflict, the *ACI that denies access always takes precedence over the ACI that grants access*. This means for example that although Peter Smith has appointed Thomas Cook as his proxy and proxies have been given access to all attributes, Thomas Cook will not be able to access all attributes of Peter Smith, because his attribute `coordinatesAccess=private` activated an ACI that denies access to his coordinates. In general, it is highly recommended to (i) minimize the number of access control rules that contain explicit denies, mainly because rules overriding can be considered hard to oversee and administer, (ii) restrict the scope of explicit grants to the smallest possible set of entries and/or attributes (i.e., adhering to the principle of parsimony regarding data access), and (iii) minimize the number and complexity of ACIs in general[58]. For more information on designing and implementing security policies, we refer to the online documentation of iPlanet Directory Server [iPlanet, 2000b], to Howes et al. [1999], and to RFC 2820 [Stokes et al., 2000].

From a theoretical point of view, the access control model we presented is based on access control lists (i.e., the aforementioned ACLs), although with many provisions for role-based access control. Following Schreck [2003], role-based access control seems especially promising for user modeling purposes, since it allows for the implementation of a variety of established security policies (for more information, we refer to Schreck [2003]). This is also demonstrated by Howes et al. [1999], who present and discuss several case studies, where role-based access policies have been implemented using the LDAP-based access control model we presented earlier.

6 User Modeling Server Architecture

In this chapter, we start with a brief presentation of a few basic principles that guided our design and implementation work. Against this background, we describe the generic architecture we developed for our user modeling server. We restrict our presentation to an overview, since we describe in greater detail an instantiation of our generic server architecture in Chapters 7 and 8. Subsequent to that, we motivate our decision for employing the commercial product iPlanet Directory Server as a foundation for our user modeling server.

In the last part of this chapter, we explore several present (and future) avenues for user modeling and the contribution our user modeling server can make to them. Thereby, we draw on advanced server facilities like (i) meta-directory facilities for integrating heterogeneous sources of user information, (ii) LDAP's distribution and replication facilities for supporting distributed user models, and (iii) LDAP's security model including iPlanet Directory Server's access control model for covering security and privacy issues (cf. Kobsa [2001b], Schreck [2003], and Kobsa and Schreck [2003]).

6.1 Overview of Server Architecture

We designed our user modeling server on top of an existing LDAP server as a 'cooperative network application' (cf. Mohr [1999]), i.e. an application that is built of autonomous components that loosely cooperate in order to establish the overall user modeling service. Each component can be transparently accessed, irrespective of network system, operating system, and programming language issues. This is accomplished by designing components as distributed objects that communicate via CORBA and LDAP (see Chapter 5.3). When

[58] RFC 2820 [Stokes et al., 2000] states for example: "Usability is a security issue, not just a nice design goal and requirement. If it is impossible to set and manage a policy for a secure situation that a human can understand, then what was set up will probably be non-secure. We all need to think of usability as a functional security requirement."

designing our components, we adhered to established design principles like the following (for more information, we refer to Orfali et al. [1994] and Mohr [1999]):

- *Coarse-grained components* that follow the notion of services. Choosing an appropriate level of granularity can be considered of paramount importance for a component-based application. In our user modeling domain, server functionality like collaborative filtering and domain-based inferences (i.e., inferences that take advantage of a domain taxonomy of interests and preferences) can be assumed to yield useful services at an appropriate level of abstraction. For more information on the mentioned services and related components, we refer to Chapters 8.5 and 8.6.
- *Encapsulate legacy systems* with a component wrapper in order to hide implementation details. This approach can be assumed to increase flexibility, since it simplifies the substitution of legacy systems during the lifetime of an application. Within our user modeling server, we wrapped iPlanet Directory Server with a component interface in order to hide this particular implementation detail from other components. From their point of view, the directory component provides dedicated services that are compliant with LDAP standards and, in case of no standard being available (e.g., for access control model), with best practices in this area (e.g., the fine-grained access control model offered by iPlanet Directory Server). For more information on our directory interface, we refer to Chapter 8.3.
- *Publicly accessible component interfaces* allow clients and servers to easily invoke methods of other components *and meta-data* enable them to handle input data in a sovereign manner (e.g., as opposed to the 'expectation-based' interpretation of incoming data in cases where no such information is available). Using XML, each discrete data element can be marked with a label that represents meta-information (e.g., that a certain number is a customer id). Moreover, a set of delimiters marks the respective begin and end of a data element (e.g. `<CustomerID>13 </CustomerID>`). For further examples and case studies, we refer to Mohr [1999]. Regarding our component interfaces, we rely on CORBA's static and dynamic invocation interfaces. CORBA maintains all object interface definitions in a dedicated '*Interface Repository*'. All CORBA objects can access this repository via dedicated APIs (for an overview of this facility, we refer e.g. to Orfali et al. [1994]).
- *Dynamic location of components at runtime* contributes to location transparency, which, in turn, leads to an increased flexibility of the overall application system. Dynamic location allows components, e.g., to be transparently migrated (or to autonomously migrate) from one computer to another (cf. the notion of user model agents [Kobsa, 2001a]). The dynamic location of components is motivated by (i) the administrator's and/or the system's ability to launch a variable number of component instances and balance the overall workload between them in order to improve the overall quality of service (e.g., fewer and/or smaller delays in service provision) and (ii) the capability to dynamically create customized services (e.g., one service acquiring assumptions about users' interests through rather quick learning and another service acquiring assumptions about users' knowledge through rather slow learning). All components of our user modeling server locate another at runtime by taking advantage of CORBA's respective facilities.
- *Components provide their service on a transient basis*. In order to cater to this, components should maintain only information that is absolutely necessary for

establishing their service. Moreover, they should rely as little as possible on the availability of other components. Adhering to these principles leads to a robust application system that degrades gracefully in case of single components being temporarily not available. If a learning component in our user modeling server becomes unavailable, e.g., then the directory component persistently buffers the input data for this component in a specific part of the usage model. As soon as this learning component becomes available again, it retrieves those data, processes them, and continues providing its service. Apart from the outage of this single service, no other service is affected and no data get lost. For more information on such a learning component, we refer to Chapter 8.4.

Against the background of these design principles and the requirements we presented in Chapters 2 and 3, we developed our user modeling server as a directory server that is complemented by several 'pluggable' user modeling components. Figure 6-1 depicts an overview of this architecture.

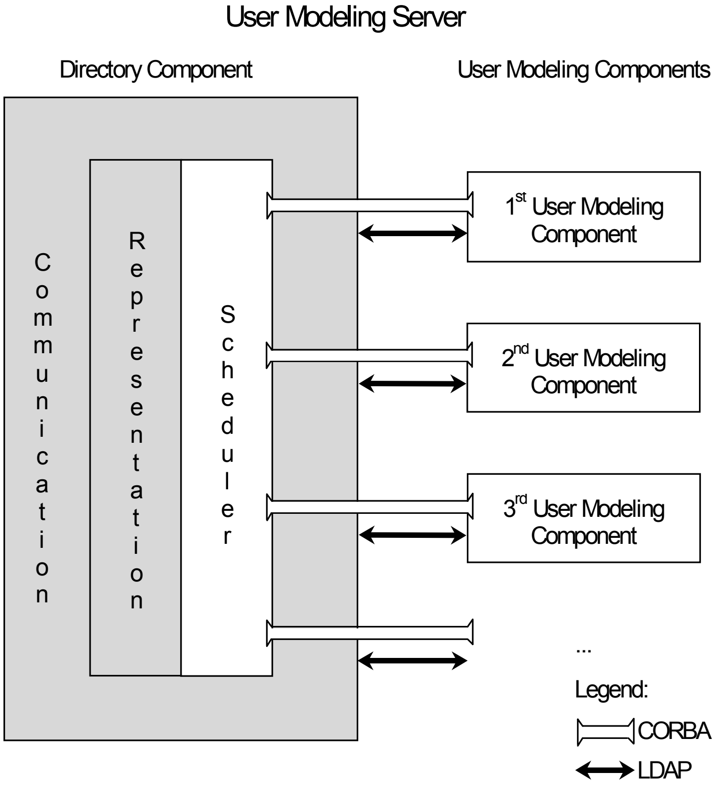


Figure 6-1: Overview generic server architecture

The left side of Figure 6-1 shows the *Directory Component* comprising the sub-systems *Communication*, *Representation*, and *Scheduler*. The Communication sub-system is responsible for managing communication with external clients of the user modeling server (i.e., user-adaptive applications) and internal clients of the Directory Component (i.e., User Modeling Components). The Representation sub-system is in charge of managing directory content (i.e., mainly user-related information). Main tasks of the Scheduler are (i) to wrap the LDAP server with a component interface and (ii) to mediate between the different sub-systems and components of the User Modeling Server. On the right side of Figure 6-1, we see several *User Modeling Components*. Each of these components performs a dedicated user modeling task (e.g., collaborative filtering, domain-based inferences). Because of their specificity, we do not elaborate them further in this overview. For more information on these components, we refer to Chapters 7 and 8.

Directory Component and User Modeling Components communicate via *CORBA* and *LDAP*. These two orthogonal communication layers are used at runtime as follows:

- The CORBA-based software bus is used for the uni-directional distribution of events and associated data from the Directory Component to User Modeling Components (e.g., an event that communicates the insertion of an interest assumption into the user model of Peter Smith). Components can register filter instructions with the Directory Component in order to control the dissemination of events and associated data on an individual basis. Event filter instructions are stored in plain text format in the directory and can be updated at any time. For more information on event filter instructions and related processing issues, we refer to Chapters 8.2.4 and 8.3, respectively. From a theoretical point of view, this communication resembles a 'filtered broadcast', a combination of the standard paradigms 'filters' and 'broadcast algorithms' for process interaction in distributed programming environments (cf. Andrews [1991]).
- LDAP is employed by User Modeling Components for accessing and manipulating information that is hosted by the Directory Component (e.g., retrieving a specific piece of information from the user model of Peter Smith).

This communication infrastructure does not mandate a specific distribution topography; components can be flexibly distributed across a network of computers, e.g., according to available computing resources. The separation of event handling and information access on different layers provides for maximum flexibility. This allows, e.g., to replace the LDAP-based information management with an SQL-based one, while still maintaining the CORBA-based communication layer.

6.2 Selection of Server Foundation

The selection of an LDAP server product as a basis for our Directory Component was based on the following requirements:

- The server requirements we elaborated in Chapter 2, including (i) multi-user synchronization, (ii) transaction management, (iii) query and manipulation language, (iv) persistency, (v) integrity, and (vi) access control.
- The directory server requirements we adopted from Howes et al. [1999] and (to a less extent from) Wilcox [1999]. These requirements include (i) support for LDAP core features (see Chapter 5), (ii) performance and scalability, (iii) security and access control, (iv) standards conformance, (v) interoperability, and (vi) flexibility and extensibility.

Before applying these requirements to directory server products, we fused them into a single catalog. We discarded the requirement multi-user synchronization, since this can be regarded a 'sine qua non' for directory server products. Transaction management was retained as a requirement for reasons we already explained in Chapter 2. Query and manipulation language was mapped to support for LDAP core features. This seems reasonable, given our aim of selecting one out of several LDAP server products. Persistency and integrity have been retained due to their paramount importance for a server product. Finally, access control has been discarded, since it is already contained in the target catalog. As opposed to the requirement query and manipulation language, the access control and security facilities offered by today's LDAP directory servers clearly excel those provided by databases (compare, e.g., SQL's grant and revoke operations with iPlanet Directory Server's fine-grained access control model we presented in Chapter 5.4).

Our final catalog contained the following requirements: (i) support for LDAP core features, (ii) transaction management, (iii) persistency, (iv) integrity, (v) performance and scalability, (vi) security and access control, (vii) standards conformance, (viii) interoperability, and (ix) flexibility and extensibility[59].

We started our evaluation of directory server products in the first quarter of 1999. At that time, only a few native LDAP directory server products were commercially available (for a list of proprietary and X.500-based directory server products that offered an LDAP gateway at that time, we refer to Howes et al. [1999]). Our market survey revealed eight products, all of them being listed in Table 6-1 along with some of their key features. The abbreviations 'v2' and 'v3' used in this table refer to the respective LDAP language versions.

Selecting an appropriate directory server product from that list was fairly easy. We started focusing on those products that provide support for LDAP core features, especially LDAP version 3 (for a motivation of this decision, we refer to Chapter 5). Applying this requirement, we could exclude four products that support LDAP v2, either with or without some provisions for LDAP v3 extensions. From the remaining four products, we discarded Active Directory Server from Microsoft, because it was not available (i.e. shipping) at that time. When comparing the remaining three products, iPlanet's Directory Server clearly excels all others (although to a different degree, as we discuss in Chapter 10). Therefore, we selected this directory server as a basis for our user modeling server.

[59] The order in which requirements are listed does not mandate any priority.

Server Name	Vendor	Key features
Active Directory Server [Microsoft, 2000c]	Microsoft	LDAP v3 support; sophisticated replication features; support for dynamic entries
DSSeries [IBM, 2000c]	IBM	LDAP v2 support; partial support for LDAP v3; security through SSL; replication features
Innosoft Distributed Directory Server (IDDS) [Innosoft, 2000]	Innosoft	LDAP v3 support; standards compliance; security through TLS
Internet Directory Server (IDS) [Lucent, 2000]	Lucent Technologies	LDAP v3 support; standards compliance; security through SSL; replication, dynamic entries; performance; scalability
iPlanet Directory Server [iPlanet, 2000b]	iPlanet (formerly Netscape, now Sun)[54]	LDAP v3 support (with roots back to SLAPD); standards compliance; security through SSL and SASL; fine-grained access control; replication; performance; scalability; multi-language support; open architecture including a virtual directory facility[60]
OpenLDAP [2000]	OpenLDAP	LDAP v2 support (based on SLAPD, see below); partial support for LDAP v3; *open source*, freely available
SLAPD [University of Michigan, 2000b]	University of Michigan and others	LDAP v2 support; partial support for LDAP v3; freely available
Sun Directory Services [Sun, 2000c]	Sun	LDAP v2 support

Table 6-1: Key features of native LDAP servers (based on Howes et al. [1999])

Today the evaluation would not be as clear-cut, since the number and maturity of directory server products considerably increased since then. Especially the open source movement

[60] A virtual directory is a directory that holds no data. It translates and reroutes LDAP calls from clients to other data stores (e.g., relational database management systems), collects results, and passes them back to its clients. To the outside world, the virtual directory appears like any other directory.

gained considerable momentum since then. This is demonstrated by the freely available OpenLDAP server, which can be considered a challenging alternative to many commercial directory server products (see also Wilcox [1999] and Shukla and Deshpande [2000]).

6.3 Support for Advanced User Modeling Scenarios

The aim of this sub-chapter is to demonstrate the support our server can offer for more advanced present and (potential) future user modeling scenarios. We also deemed this presentation necessary, since the custom user modeling server we developed for Deep Map (see Chapter 7) does not take advantage of some of the more interesting facilities of our server. In this vein, we discuss the following three scenarios:

- '*Monoatomic user modeling*', which focuses on the physical integration of distributed and heterogeneous information about a client within a company.
- '*Polyatomic user modeling*', which demonstrates how distributed and homogeneous user-related information can be virtually fused in a single user model.
- '*Secure and Private User Modeling*', which focuses on selected security and privacy issues.

In the following, we sketch each scenario and briefly present selected facilities of our server against this background. Thereby, we leave out other server facilities that may be also relevant or even mandatory, mainly for reasons of brevity. A further elaboration of these scenarios goes beyond the scope of this thesis and is left for future research in this area.

6.3.1 Monoatomic User Modeling

Integrating client-related information that is scattered across an enterprise in a single repository can be regarded of paramount importance for businesses (see Chapter 1.2). This allows for leveraging many of the advantages of a central source of client information (e.g., up-to-date information for holistic personalization, synergistic effects with respect to acquisition and usage, support for the holistic design, acquisition, and maintenance of client information), while still preserving important benefits of their decentralized management (e.g., high quality, up-to-dateness, low maintenance costs). We already covered these issues in Chapters 1.2 and 1.3.

Figure 6-2 depicts a scenario, where meta-directory facilities enable our User Modeling Server to *physically integrate* two sources of client-related information: a Marketing Database and an ERP (i.e. Enterprise Resource Planning) System from vendors like SAP [2001] or Baan [2001]. The Marketing Database is depicted at the left bottom of our figure and comprises two sections. The first section contains information about individual clients' names, interests, CLTV (i.e., Customer LifeTime Value) forecasts (see Chapter 1.2), and clients' association to available client segmentations. The second section encompasses client segmentations, which have been acquired by employing clustering and classification techniques on the aforementioned clients' interests (for more information on such software products, we refer e.g. to Woods and Kyral [1997]). The ERP System is depicted at the right bottom and contains administrative information about clients' names, addresses, accounts, invoices, and some statistics including clients' (total) turnover. The User Modeling Server is depicted on the top. It maintains information about client segmentations and information about individual clients' names, passwords, addresses, and their association to available client segmentations. Moreover, the User Modeling Server acquires and maintains assumptions about clients' interests and preferences (see Chapter 8 for more information on this issue).

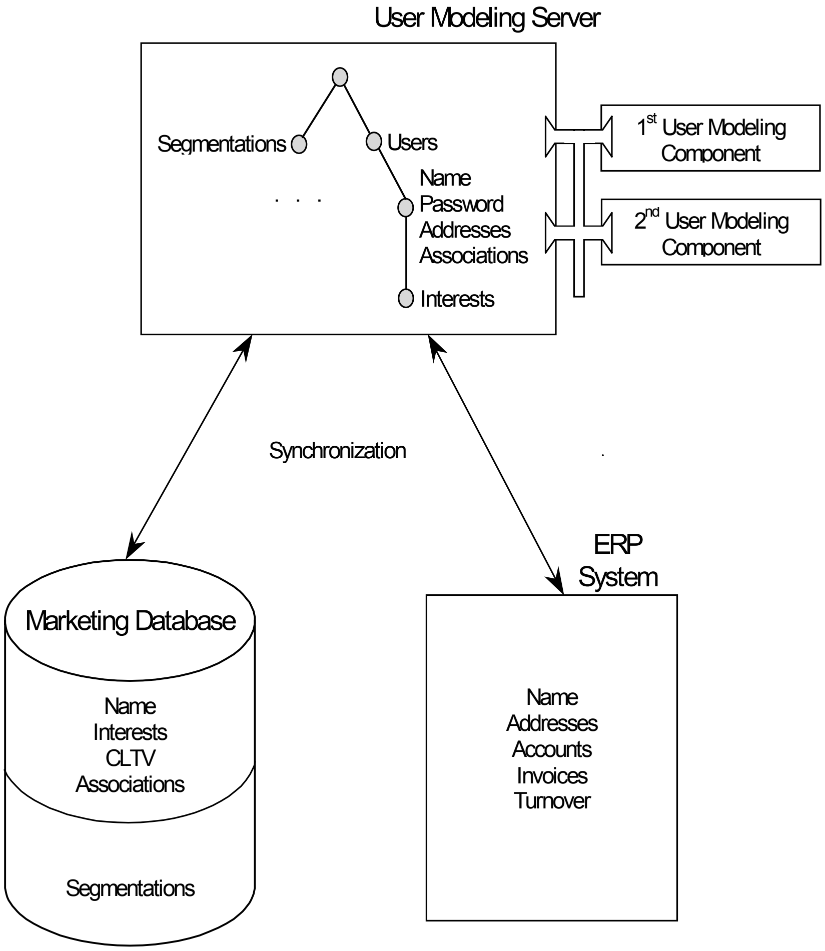


Figure 6-2: Scenario monoatomic user modeling

Looking at the client information available in the different repositories, we see that some information seems to be redundant (e.g., clients' interests, addresses, and client segmentations). In fact, this redundancy is established by directory synchronization software from vendors like Persistent [2000] and Critical Path [2000]. Employing synchronization software turns our User Modeling Server into a meta-directory, a single (physical) source of client-related information.

In order to achieve this, the synchronization software mirrors selected client information between the different repositories and henceforth keeps these mirrors in sync with their original sources. The administrative responsibilities (including the question of ownership) regarding client information can be assumed in our scenario as follows:

- User modeling server: clients' names, passwords, interests.
- Marketing Database: CLTV forecasts, client segmentations, and their association to individual clients.
- ERP System: clients' names, addresses, accounts, invoices, and turnover.

Assuming certain requirements regarding frequency, direction, and security of the synchronization process, we can subsequently configure the synchronization software for establishing the necessary information flows between the different repositories (e.g., client segmentations and clients' respective associations from the Marketing Database to the User Modeling Server, clients' interests from the User Modeling Server to the Marketing Database). As soon as these information flows are established, clients' interests acquired and maintained by the User Modeling Server can be used by the marketing department, e.g. for assessing individual client's lifetime value (e.g., by following a CLTV approach like the one proposed by Cooperstein et al. [1999]). And vice versa, client segmentations can be used by the User Modeling Server for supporting its learning processes (see Chapter 8.5 for an example). In our scenario, we used clients' names as a key for joining the different data sources, since they are contained in all repositories[61].

In general, synchronization products allow for establishing the following types of synchronization schemes:

- *One-way synchronization* periodically updates a target repository with information from a source[62]. The flow of information is uni-directional, from the source to the target. Update can be either total or incremental. In case of a total update, all information in the target is deleted and replaced with the information from the source. An incremental update applies only those changes to the target that occurred in the source since the last synchronization. The advantages and disadvantages of both approaches are apparent: total updates can require a substantial amount of computing and networking resources, but do not mandate sophisticated change tracking mechanisms in repositories; and vice versa for incremental updates. Examples of one-way synchronization in our scenario include (i) the update of client segmentations and clients' respective membership in the User Modeling Server from the Marketing Database and (ii) the update of clients' interests in the Marketing Database from the User Modeling Server.
- *Two-way synchronization* periodically propagates changes among several repositories. Such an approach provides maximum organizational flexibility since it allows a piece of information to be changed in any of the synchronized repositories. In our scenario for example, clients may change their addresses via a Web interface in the User Modeling Server and call center staff can update clients' addresses in the User Modeling Server as well as in the ERP system. Irrespective of the source that has been changed, the synchronization software takes care of keeping all repositories in sync (i.e., the User

[61] We leave aside for the moment the more practical problems of such an approach (e.g., duplicate names). For more information on this topic, we refer to Howes et al. [1999].

[62] One-way synchronization may happen only once (e.g., when initially populating the User Modeling Server with clients' names and addresses from the ERP system).

Modeling Server and the ERP system in our example). This advantage, however, comes at the price of increased computational complexity. Update conflicts may occur, when the repositories and the synchronization software do not support tight synchronization schemes, as provided e.g. by a two-phase commit protocol (see Chapter 2.2.2).

- *N-way join synchronization* allows for periodically updating a target with information that is obtained by joining several sources. In our example, clients' addresses from the ERP system and clients' associations to segmentations from the Marketing Database can be joined via the common name attribute, before propagating the result (i.e., a set of information triples, each of them comprising a client's name, addresses, and segmentation associations) to the User Modeling Server. The efforts necessary for establishing an n-way join synchronization can be regarded considerable, mainly depending on the amount of information to be joined and the respective facilities offered by the repositories involved in the synchronization effort.

The synchronization schemes that are at the disposal of system designers are mainly determined by the respective facilities offered by synchronization products. The various technical and organizational aspects of a meta-directory require in any case careful planning. At runtime, a meta-directory can be assumed to require a considerable amount of computing and network resources, mainly depending on the direction, frequency, and complexity of the synchronization processes. For more information on meta-directories, related case studies, and pointers to additional resources, we refer to Howes et al. [1999].

6.3.2 Polyatomic User Modeling

An alternative approach for leveraging the advantages of centralized and decentralized management of user-related information is their *virtual integration* into a single polyatomic source. We exemplify this in the following by linking an LDAP-based repository and two models of user-related information through referrals (see Chapter 5.2)[63]. Compared to the scenario presented in the last sub-chapter, the main difference is that integration is not established through moving information between sources and targets, but through linking information in the repositories involved in the synchronization process. In other words, proactive moving of information between repositories is traded in for retrieving information at runtime. We assume that linking can be regarded a sine qua non for many integration scenarios, since this does not raise problems of information ownership and administrative responsibility. On a technical level, linking does not require additional software products. Referrals can be regarded the mechanism of choice regarding linking implementation, since they are a part of the LDAP v3 specification [Wahl et al., 1997a].

Figure 6-3 depicts a scenario, where User Modeling Server A *virtually integrates* two external sources of client-related information: a Domain Server and a second User Modeling Server (i.e., Server B). The Domain Server depicted at the left bottom is owned by a company, which is specialized in leasing computer hardware. Their Domain Server hosts plenty of information about machines owned by that company, but maintains at the same time information about users, devices, and applications. This server is based on

[63] We thereby assume that the different sources of our polyatomic user model comply with the four LDAP models we introduced in Chapter 5. Apart from this requirement, these sources can be quite heterogeneous, e.g. regarding their representation system, content, and physical location.

Microsoft's Active Directory[64] [Microsoft, 2000c]. The second source of information is User Modeling Server B, which is depicted at the right bottom of our figure. User Modeling Server B is employed by an online book store for acquiring and maintaining clients' interests and preferences regarding books. A second server (i.e., User Modeling Server A) is employed by this book store for acquiring and maintaining employees' interests and preferences regarding information available on their intranet (e.g., product news, job offerings).

Against this background, employees can ask their IT department for linking their various user models with the user model hosted by Server A. If this is feasible from a technical and organizational point of view (e.g., sources are based on LDAP, at least read access is granted), an administrator adds appropriate referrals to an employee's user model. In our case, two referrals are added to the user model hosted by Server A. These referrals point to the machine model hosted by the Domain Server and the interest and preferences model hosted by User Modeling Server B. From a technical point of view, these referrals are simple URLs[65] like the following:

```
ldap://ad.company.com:389/cn=Laptop123,ou=Machines,o=company.com
```

As soon as these links are established, an employee can use an arbitrary (Web-based) directory application (e.g., Publisher from Oblix [2000]) or directory browser[66] (see Chapter 8.2) for inspecting these models, starting from the model hosted by Server A. If appropriately authorized, users can also block, rectify, and erase directory entries in these models (see Kobsa [2001b], who claims these user rights from a privacy point of view).

Moreover, User Modeling Server A can take advantage of this additional information as well. Its learning components, if appropriately authorized and configured, can use the machine information as well as users' interests and preferences regarding books, e.g. for finding groups of like-minded users (e.g., users that utilize a laptop from the same vendor, users that share a common interest in SF literature). For more information on such a learning component, we refer to Chapter 8.5.

[64] Active Directory is an integral part of the Windows 2000 operating system.

[65] LDAP URLs (i.e. Uniform Resource Locator) are defined in RFC 2255 [Howes and Smith, 1997a].

[66] Most directory browsers seem to offer low-level access to directory information only and are therefore hardly appropriate for end users.

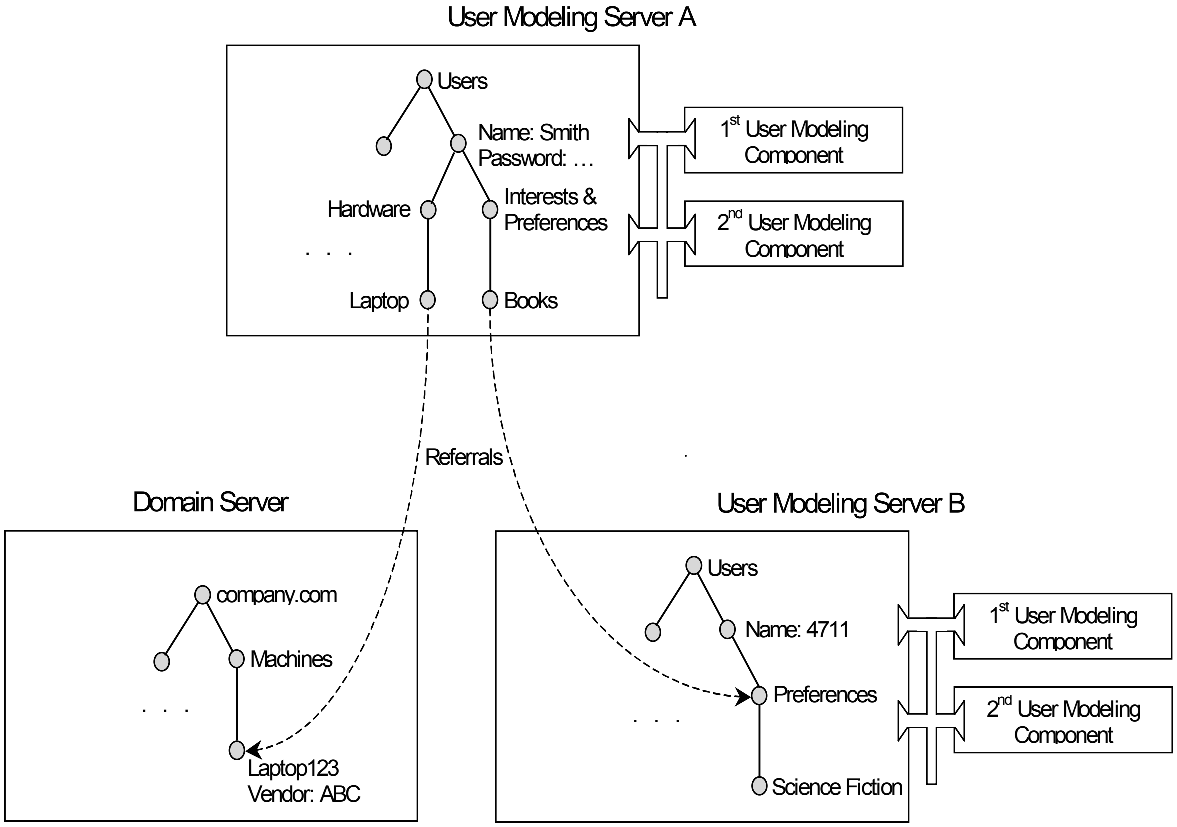


Figure 6-3: Scenario polyatomic user modeling

From a user's point of view, different levels of transparency can be provided regarding referrals. In the simplest case, neither the LDAP server, nor the LDAP client, nor the directory application automatically chases a referral returned by User Modeling Server A. In this case, it is left to the user to point her directory application to the address indicated by the referral and to provide appropriate authentication credentials to the new repository. On the other extreme of the transparency continuum, the whole polyatomic user model appears to the user as a single uniform repository of user-related information. All low-level details including referrals are hidden from the user. This is accomplished by either the server, the LDAP client, or the directory application automatically chasing referrals, handling authentication, and appropriately assembling the result sets returned by the different servers.

Although this uniformity can be considered appropriate for some real-world scenarios (e.g., where users' convenience seems important and the involved repositories can be considered trustworthy), such an approach can severely affect at the same time systems' security and users' privacy. Regarding security for example, it is of paramount importance that only a few people (e.g., selected administrators) are allowed to maintain referrals. Many (LDAP) clients automatically chase referrals and simply resubmit their credentials to the server they are referred to (i.e., in case of simple authentication, they resubmit their passwords as plain text over the network to the server). If an attacker is able to place a referral to herself into the directory, then she can record the credentials of all those clients that automatically chase this referral. Subsequently, the attacker can use these credentials for impostering as legitimate clients. This replay of credentials will become ineffective, however, as soon as SSL or Kerberos are used for individually signing each message exchanged between client and server (see Chapter 5.4)[67]. Regarding privacy, an important motivation to not automatically chase referrals is to communicate to the user the current and future server environment. This enables a user to assess a server's trustworthiness and adapt her interaction accordingly (i.e., refuse or chase a referral).

In conclusion, the degree of linking transparency provided to end users requires careful consideration for a polyatomic user model. Given the potential implications for systems' security and users' privacy, completely hiding referrals from end users seems hardly desirable, since this precludes their awareness regarding the server environment. This is mandatory, however, for enabling users to act according to the presumed trustworthiness of their computing environment.

6.3.3 Secure and Private User Modeling

In Chapter 5.4, we discussed the following security and privacy threats in user modeling:

- *Unauthorized access*, e.g. through Trojan horses, connection hijacking, network sniffing, and backdoor access.
- *Unauthorized tampering*, e.g. through 'man in the middle' attacks and various forms of masquerading.
- *Non-accessibility*, e.g. through illegitimate consumption of directory services and hardware resources.

In the remainder of this sub-chapter, we focus on the first two categories. For safeguarding our server against non-accessibility attacks, we refer to Chapter 5.4, where we briefly presented auditing and resource control facilities.

In the following scenario, we showcase selected security and privacy threats to user modeling (cf. Kobsa [2000]). Figure 6-4 depicts two users (i.e., User 1 and User 2) that take advantage of two user modeling servers (i.e., User Modeling Server X and User Modeling Server Y) via several Web-based applications (i.e., User-Adaptive Application 1 to 4). All connections between the applications shown can be subject to unauthorized access and tampering attacks. This is exemplified for the connections between User 1 and User Modeling Server X. Between User 1 and User Interface A, for example, a Trojan horse can imitate a login screen and try to steal the credentials of User 1. The connection between User Interface A and User-Adaptive Application 1 might be subject to network sniffing and the connection between User-Adaptive Application 1 and User Modeling Server X can be jeopardized by hijacking attempts. Shaded boxes depict various imposterers that try to masquerade as legitimate applications (i.e., user interfaces, user-adaptive applications, and user modeling servers).

[67] SSL and Kerberos attach to each message an authentication code that contains a sequence number.

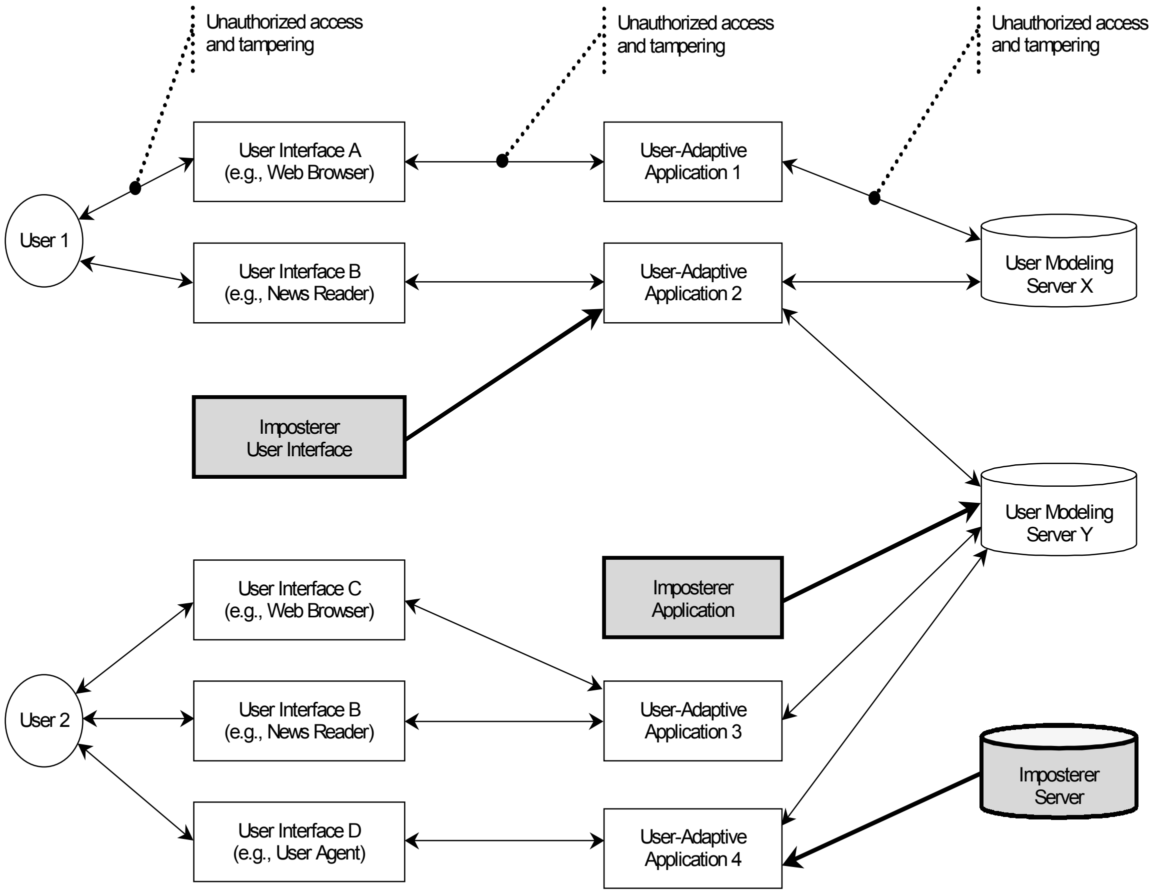

Unauthorized access and tampering
Unauthorized access and tampering
Unauthorized access and tampering
User 1
User Interface A (e.g., Web Browser)
User Interface B (e.g., News Reader)
User-Adaptive Application 1
User-Adaptive Application 2
User Modeling Server X
Imposterer User Interface
User Modeling Server Y
User Interface C (e.g., Web Browser)
Imposterer Application
User 2
User Interface B (e.g., News Reader)
User-Adaptive Application 3
Imposterer Server
User Interface D (e.g., User Agent)
User-Adaptive Application 4

Figure 6-4: Security and privacy threats in user modeling (based on Kobsa [2000])

In order to safeguard client-server communication against these security threats[68], our user modeling server offers support for *authentication*, *signing*, and *encryption*. This is established by (mainly) relying on SSL and, in any way, on established or proposed Internet standards (see Chapter 5.4 for more information on directory security). Custom extensions (e.g., a private key authentication system like Kerberos, an encryption algorithm like MD-5) can be integrated into our server via the SASL framework. Additional facilities of our server include support for *access control*, *auditing*, and *resource control*. These iPlanet Directory Server facilities can be regarded best practice for directory servers [Howes et al., 1999; Mohr, 1999; Wilcox, 1999; Shukla and Deshpande, 2000]. For more information on these facilities, we refer to Chapter 5.4. For more information on respective facilities of our CORBA-based communication layer, we refer to Chapter 8 and to the VisiBroker documentation from Inprise [2000].

Regarding users' privacy, Schreck [2003] points out that the sensitivity of a piece of user-related information mainly depends on its associability with an individual person. This is

[68] It is important to note that the security facilities provided by our user modeling server can not safeguard against all threats we presented. Additional precautions have to be taken (e.g., antivirus software, organizational regulations) for protecting against threats like Trojan horses on users' desktops.

also reflected by many privacy regulations[69], guidelines, policies, and best practices in this area focusing on information that can be associated with individuals. The stronger a piece of information can be associated with an individual, the more restrictions are applied to its processing, storing, and sharing with third parties. The restrictions applied are driven by two basic principles: *parsimony* and *purpose-specificity* (cf. Kobsa [2000]). Parsimony restricts the amount of user-related information according to a dedicated purpose (e.g., paving the way for a contract, fulfillment of a contract). This implies that user-related information must not be stored longer than necessary. Purpose-specificity restricts the storage and processing of user-related information to a dedicated purpose and prohibits its exploitation for orthogonal purposes. Additional restrictions may apply in case of sensitive user-related information (e.g., users' case histories, religious or philosophical beliefs, racial or ethnic origins)[70].

As opposed to that, many real-world personalization environments seem to violate the aforementioned principles, since

- "as many data as possible is being collected and 'laid in stock'" and
- "data are exploited for purposes other than those for which they were originally collected" [Kobsa, 2000].

This is perhaps motivated by companies' investments in personalization infrastructures and related customer relationship management platforms (see Hagen et al. [1998] and Millhouse et al. [2000]). Following this, many sites expect a payback for their personalization efforts in the form of personal data (we already mentioned this in Chapter 3.2.1). However, acquisition and processing of personal data should (i) be amenable to applicable privacy regulations and/or authorized by users' voluntary and informed consent, (ii) provide users means to inspect, block, rectify, and erase personal data, (iii) employ state-of-the-art security mechanisms according to the sensitivity of the personal data, (iv) conform to users' expectations, and (v) be outweighed by the usefulness of the personalized services offered [Hagen et al., 1999].

From a personalization point of view, maintaining the relation between a piece of user-related information and an individual person is rarely necessary. Exceptions where this deems necessary include prevailing real-life personalization scenarios, where potentially sensitive information (e.g., e-mail addresses, mobile phone numbers, users' addresses) is indispensable for establishing a service (e.g., sending tailored alerts via e-mail or SMS[71], support express transactions).

[69] Privacy regulations restrict the processing, storing, and sharing of user-related information in many countries. For an overview, we refer e.g. to Kobsa [2001b] and Schreck [2003].

[70] Some of these restrictions can be waived upon explicit consent of the user. For more information, we refer to Kobsa [2000].

[71] SMS (i.e., Short Message Service) allows for asynchronously exchanging short messages in written form, e.g. via mobile phones.

The user modeling server we developed puts several facilities at the disposal of administrators and user model developers that allow them to cater to the sensitivity of user-related information. These facilities include all of the following (for more information on these facilities, we refer to Chapter 5):

- *Access control* can be used for restricting access to user-related information in a fine-grained manner (e.g., grant access to users' e-mail address only to specific applications and users, restrict access to a single user's address to a period in time).
- *Distribution* allows for segregating parts of the user model according to their potential sensitivity (e.g., demographic information about users can be stored on one server, whereas interest sub-models can be stored on other servers that may be less secure). Where necessary, referrals can be used for linking the segregated parts of a user model together.
- *Replication* allows for the regular duplication of parts of a user model from one server to another. This can be used, e.g., for maintaining potentially sensitive information about users (e.g., name, age, sex) on a secure server and to replicate potentially less sensitive information (e.g., users' interests and preferences) to a server that is open to the public.
- *Modification tracking* records all changes to a user model entry in special attributes (i.e., attributes creatorsname, createtimestamp, modifiersname, modifytimestamp). These so-called 'operational' attributes are exclusively maintained by the server for each user model entry. Clients are granted read and search access to these attributes. 'Back door' modifications, however, are prohibited, since clients (including administrators) are not able to modify operational attributes.
- *Chaining* of communication (see also Chapter 5.2) between clients and user modeling servers allows for establishing various degrees of procedural anonymity[72] and pseudonymity (cf. Schreck [2003]). All servers that support chaining (e.g., X.500 servers) and super-identification[73] (either via SSL, TLS, or a private key system like Kerberos) can be used as intermediate servers. We believe, however, that such an approach is restricted to the rather small number of personalization scenarios, where the time (including its variance) needed for delivering a message is not critical. An example of such a scenario is the assembling and sending of personalized messages via e-mail and SMS.
- Support for *dynamic contents* allows for the maintenance of potentially sensitive user information within a user model on a transient basis. Dynamic user model entries are not persistent and need to be periodically refreshed by a client application. An e-mail address for example, which is appropriately protected by access control information, is kept only as long in a user model as needed for assembling a personalized e-mail; if this service does not periodically refresh this e-mail address, then it vanishes from the user

[72] Procedural anonymity can be established for clients and servers by passing a message between them through a set of intermediaries, thereby keeping the original sender and receiver secret (e.g., by encrypting their identities on a content level). For more information, we refer to Schreck [2003].

[73] With super-identification, communication partners take advantage of an external authority for authenticating the identity of one another. The X.509 standard we mentioned earlier is an example of a super-identification service.

model). For more information on this proposed extension to LDAP version 3, we refer to RFC 2589 [Yaacovi et al., 1999].

The following scenario, which is depicted in Figure 6-5, demonstrates some of these facilities from a user's point of view. Thereby, we also revisit and integrate parts of former scenarios.

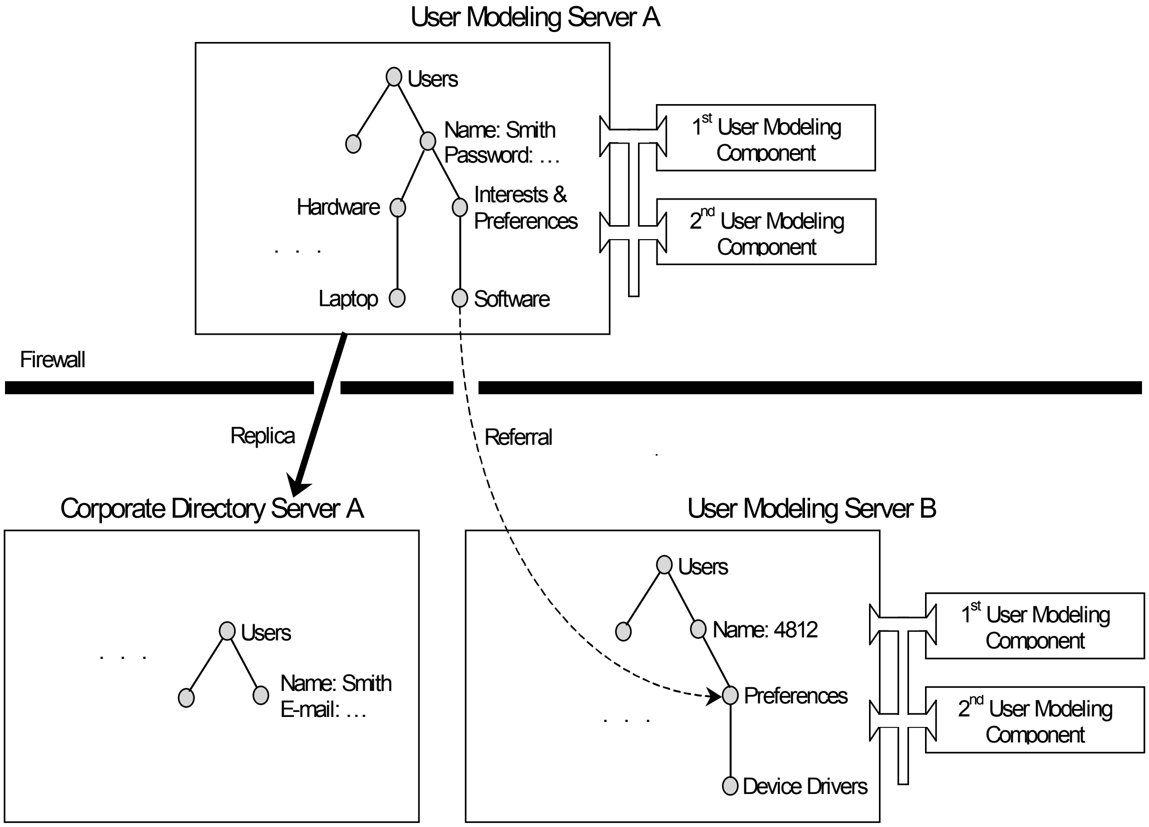


Figure 6-5: Scenario secure and private user modeling

User Modeling Server A hosts information about employees in a company. The user model of Mr. Smith, an experienced programmer, contains several demographic attributes, an interests and preferences sub-model, and information about the computer hardware currently used by him (i.e., a single Laptop). Once in a while, Mr. Smith checks in at an external online software shop which is specialized in component software. Mr. Smith remembers very well his first visit to that shop. At that time, he was offered plenty of information about their privacy policy, about the user-related information they collect, the rationale and purpose for collecting each piece of information, the personalized services they offer based on that information, etc. After a final question whether an account should be created for him or not, he decided to create one using the pseudonym '*4812*' and a simple password (at that time he wondered a little bit about their promise to provide access to their shop anyway). This account was created for him on User Modeling Server B.

Since then, each time Mr. Smith checks in, he gets personalized recommendations regarding potentially interesting Device Driver software. Mr. Smith is also offered a tool for inspecting and editing his user model. He used this tool once for checking out the systems assumptions about his interests and preferences. Since then, he knows that (i) the shop only stores his pseudonym, password, interests and preferences and that (ii) their tool for user model inspection was a bit clumsy, compared to the one he is using for maintaining the user model in his company. Therefore, he recently asked his administrator to setup a link from his local user model to that external one, thereby allowing him to inspect both models from a single starting point with his favorite tool. Since then, each time he comes across that link he gets (i) a notification that the trusted server environment is about to change and (ii) an authentication request from the external server. After entering the credentials for his pseudonym 4812, he is granted access to the external user model.

When Mr. Smith recently contacted his administrator, he also asked him to periodically replicate a few attributes (mainly Name and E-mail address) from his local user model to the corporate directory in front of the firewall, which is hosted by Corporate Directory Server A. Since then, the number of clients, colleagues, and friends asking for his (new) coordinates dramatically decreased.

In conclusion, we believe that our server exhibits facilities that enable administrators and user model developers to safeguard against many security threats in user modeling and to account for the sensitivity of user-related information. The interests and preferences models hosted by User Modeling Server B, for example, can hardly be related to individuals, since (i) they contain barely any sensitive information and (ii) the elicitation of combinations of user-related information that would facilitate deanonymization can be assumed very difficult (cf. Schreck [2003]), since the number of interests and preferences models can be assumed quite large.

Some of the security and privacy facilities of our server correspond to facilities developed by Schreck [2003]. In his work on security and privacy issues in user modeling, he proposes the following facilities:

- Regarding *authentication*, *signing*, and *encryption*, he proposes an SSL-enhanced version of KQML (see Finin et al. [1993] and Labrou and Finin [1997]) for communication between user-adaptive applications and user modeling servers.
- As regards *access control*, he proposes a role-based access control model.
- For establishing *procedural anonymity and pseudonymity*, he proposes a KQMLmix implementation. This allows for hiding senders' and receivers' identity by (i) encrypting it on the message layer and by (ii) routing messages through a set of intermediaries, where each intermediary only has knowledge about its immediate neighbor.
- Regarding *modification tracking*, he proposes a mechanism that allows for storing information about the origin of user model entries.

The most important difference between our security facilities and the ones Schreck proposes is that we focus on LDAP as a basis for our user modeling server, whereas he does not make such an assumption. His facilities are intended for a rather broad range of user modeling servers (e.g., regarding their representational and inferential capabilities).

III
User Modeling Server Implementation

7 User Modeling Server for Deep Map

In this part of our work, we demonstrate the validity of our generic server architecture by instantiating a user modeling server for the Deep Map project. In the following chapter, we briefly describe the project context, the specific user modeling requirements we found, and the architecture of the user modeling server we developed.

In a subsequent chapter, we describe the User Modeling Components that we developed for Deep Map and specifically the incorporated learning techniques that we adopted from the area of machine learning for user modeling. We argue that by integrating these User Modeling Components into a single server we can leverage several synergistic effects between these techniques and compensate for well-known deficits of individual techniques.

7.1 User Modeling in Deep Map

Deep Map [Malaka and Zipf, 2000; Deep Map, 2001] is part of a family of long-term research projects aimed at developing personal Web-based and mobile tourist guides, thereby integrating research from various areas of computer science: geo-information systems, data bases, speech input and output, multilingualism, intelligent user interfaces, knowledge representation, and user modeling (see EML [1999; 2000], Malaka [1999], and Malaka and Zipf [2000] for more details about the project aims). In order to enable Deep Map and other components to provide personalized behavior, a model of relevant characteristics of individual users (namely users' individual interests and preferences) and of user groups has to be acquired and maintained by a user modeling server (abbreviated 'UMS' in the following).

A central aim of the Deep Map sub-project 'WebGuide' [2001] is the provision of personalized tour recommendations for the city of Heidelberg that cater to an individual user's interests and preferences. WebGuide identifies geographical points of interest and computes a tour that connects these points via presumably interesting routes based on the following pieces of information:

- geographical information about Heidelberg,
- information about relevant points of interest (e.g., the Heidelberg Castle),
- information about selected means of transport (e.g., car or bike),
- individual user's interests and preferences, and
- tour restrictions specified by the user (e.g., regarding distance and duration).

Tour recommendations that meet the above requirements are then presented to the user. Figure 7-1 depicts two proposals for walking tours that were prepared for the same user by a first prototype of WebGuide. The tour proposal on the left side does not take individual user interests and preferences into account, while the proposal on the right respects particularly the user's dislike of environmental burden. Although both tours contain the same points of interest (indicated by black dots), the proposed routes between these points (depicted by bold lines) differ especially in the encircled areas. In the personalized tour, routes that are presumably problematic with respect to environmental burden (e.g., routes along streets with high traffic) are substituted by more appropriate paths (e.g., by routes through pedestrian zones, parks, and forests), if possible.

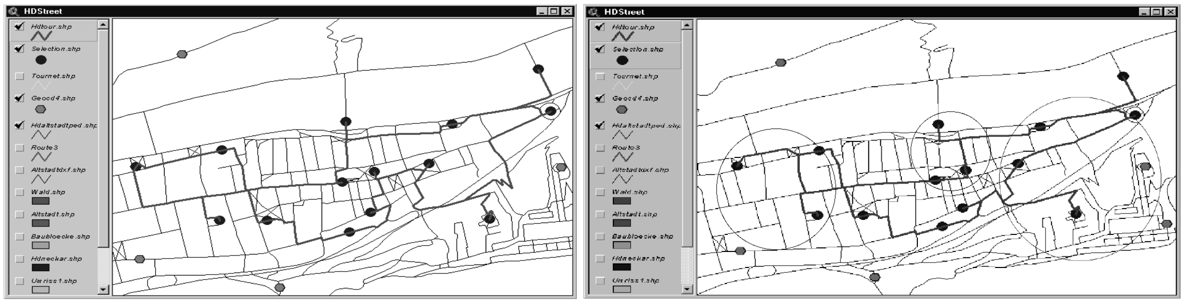


Figure 7-1: WebGuide tour proposals [EML, 1999]. Reprinted with permission.

In addition to the more generic user modeling requirements we introduced in Chapters 2 and 3, we elicited the following more specific user modeling requirements in a number of scenario-based design sessions [Carroll, 1995; 2000]:

- *User characteristics* that need to be taken into account *include interests and preferences*, *and selected demographic data* (e.g., users' age, gender, and continent of origin[74]).
- *A priori knowledge about users* (e.g., from other user-adaptive systems or smartcards that store users' interests [Fink et al., 1998]) *is generally not available*.
- The *explicit acquisition* of user information at runtime *must be restricted to a brief initial interview*. The *emphasis* must lie *on* the *implicit* (i.e. unobtrusive) *acquisition* of user interests and preferences, e.g. from user feedback, usage data, models of similar users, and inferences based e.g. on domain heuristics.
- *Adaptation should be* relatively *quick*. For instance, the provision of personalized information and services should already be possible during the user's first session with Deep Map. In subsequent sessions, the system should be able to cater to a user's changing interests and preferences.
- *Long-term user modeling* should be supported (which implies that the lifetime of individual user models in Deep Map must extend beyond a single user session).
- *Security and privacy*, and related technical implications (e.g., scrutability of user model contents by Deep Map users) should be taken into account.

Other identified requirements result from related user modeling research, or simply conform to best practice in the design of software systems:

- *Easy access* to the user modeling server should be possible from different applications, and different software and hardware platforms.
- Different *user model acquisition techniques should complement each other* and *synergistic effects between methods* should be exploited. For instance, an acquisition method that predicts user characteristics based on similarities between user profiles should be able to take interests and preferences into account that were explicitly

[74] The importance of acquiring and maintaining selected demographic data is underlined by Kaul [1999]. Her empirical work on tourism in Heidelberg revealed that the interest of tourists in 17 out of 19 touristic activities significantly depends on their age, and that 14 out of 19 interests significantly depend on their continent of origin.

provided by users, as well as interests and preferences that were implicitly acquired by other acquisition methods.

- The user modeling server must be *open* with respect to
  - new or evolving user modeling requirements and their implications for the acquisition and representation of user models, for learning methods, and for inferences on the basis of user models,
  - external data sources about users that may become available in the future (e.g., P3P profiles [Reagle and Cranor, 1999]), and
  - tools for user model analysis (e.g., visualization tools, data mining tools) and associated interface standards like ODBC.
- User models must be *upwards compatible* between existing and future Deep Map prototypes.
- The *quality of service* of the user modeling system is very important (e.g., its performance, responsiveness, and robustness).
- The *management of arbitrary information about Deep Map components* (their configuration, location, etc.) should be possible within the user modeling system. This enables especially mobile and location-aware applications of Deep Map to flexibly adapt their services to the available computing resources (locally available resources on mobile devices are usually much more limited than server resources through a network).
- The user modeling system should comply as much as possible with existing and emerging de jure and de facto *standards* (e.g., from organizations like ISO and W3C).

These requirements substantially influence the architecture and the properties of our user modeling server along with the more general requirements we presented in Chapters 2 and 3. The user modeling demands we collected in Deep Map are, however, not unusual, but fairly typical for user-adaptive information systems that are being used in short interactions only.

7.2 Overview of Server Architecture

Figure 7-2 depicts the architecture of the UMS for Deep Map, which is an instantiation of the generic architecture we presented in Chapter 6.1. Several External Clients are shown on the left side (e.g., Deep Map Agents, Browsers). The remaining part of this figure delineates the UMS, which comprises a Directory Component (in the middle of the figure) and a set of User Modeling Components (on the right). Sub-systems of the Directory Component are depicted in the middle of the figure (e.g., Communication, Scheduler). User Modeling Components of the UMS are depicted on the right (e.g., User Learning). External communication with the UMS is established via standard protocols (e.g., LDAP, ODBC)[75]. Within the UMS, there are two levels of communication that are based on CORBA and LDAP. In the remainder of this sub-chapter, we briefly describe each group of external clients, UMS sub-systems and components.

[75] Not all communication links are depicted, many more are possible (e.g., Browsers accessing the UMS via ODBC).

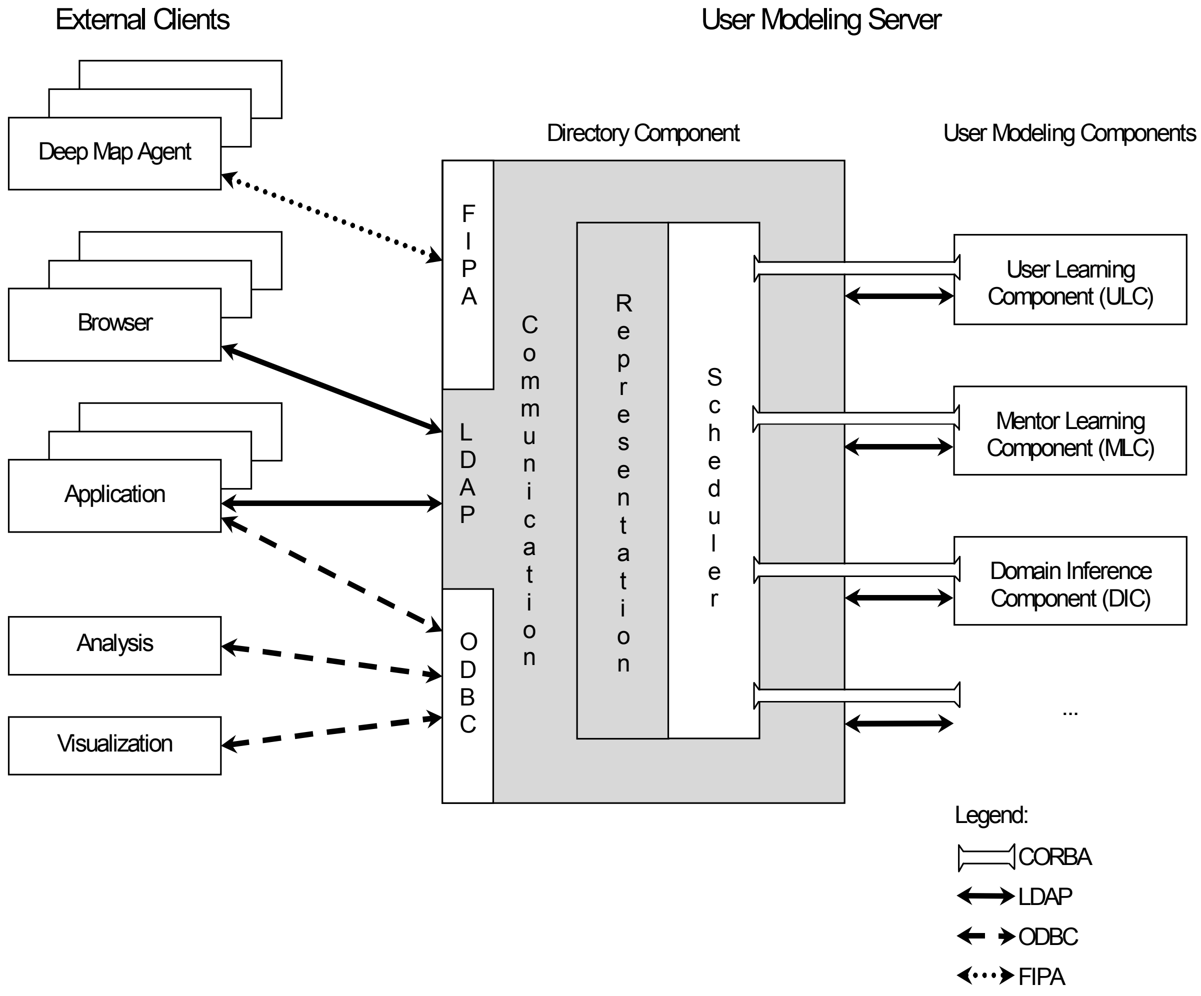


Figure 7-2: User Modeling Server architecture for Deep Map

Deep Map Agents are autonomous software components that provide tour recommendations, analyze spoken input and generate speech output, interface to the World Wide Web, etc. Deep Map Agents communicate to the UMS various data about the user's interaction with the system and user characteristics. Examples of the user's interaction include user requests for information on environmental burden in Heidelberg, or gothic works of art in a specific church. An example of user characteristics is a user's explicit statement in an initial interview that she is heavily interested in information about churches. Conversely, Deep Map Agents query the UMS for user characteristics (e.g., retrieve from the model of the current user an assessment regarding her interest in information about the Early Middle Ages, or retrieve all interests and preferences with a probability greater than 0.7) and for information about the system environment (e.g., which User Modeling Components are currently available, or where Deep Map sub-systems are currently located). From an architectural point of view, Deep Map Agents loosely adhere to the agent specifications released by FIPA (Federation for Intelligent Physical Agents), especially to FIPA [1998a; 1998b][76].

System administrators carry out user model management tasks (e.g., configure UMS components, manage access control, maintain assumptions in models of individual users) via an LDAP Browser. Within its interface, standard LDAP operations (e.g., add, update, and remove entries) can be applied to the models hosted by the UMS. Additional tools can be used for administering the UMS, e.g. commercially available access control management tools [Baltimore, 2001; Netegrity 2001].

Users and applications that take advantage of the UMS can inspect user model contents (e.g., search for a specific user, list individual interests and preferences) using a variety of LDAP-compliant Applications, including the following ones:

- Web browsers (e.g., Microsoft Internet Explorer [Microsoft, 2000a], Netscape Communicator [Netscape, 2000a]);
- e-mail clients (e.g., Eudora from Qualcomm [2000]);
- standard LDAP browsers (e.g., Globus browser [Gawor, 1999], Active Directory Browser from Microsoft [2000b]), which allow for browsing, modifying, and searching user model contents (e.g., adding an interest in museums with a probability of 0.9);
- HTML-based (end) user interfaces for LDAP servers (e.g., from Oblix [2000]);
- server applications that cooperate with LDAP servers (e.g., Web servers and security servers from iPlanet [2000a], Web development environments like 'WebObjects' from Apple [2000], e-commerce application environments like One-To-One from BroadVision [2000], commercially available user modeling servers like Personalization Server from ATG [2000]).

Application designers may decide, however, to provide a custom UMS interface in their applications. Custom interfaces may, e.g., restrict browsing to dedicated parts of a user model. In order to develop such an interface, programmers can take advantage of a variety of (mostly) freely available LDAP SDKs and interface components, e.g. the LDAP C SDK from the University of Michigan [2000a], various Directory SDKs from Netscape [2000b], LDAP-X from Boldon James [2000], JNDI (Java Naming and Directory Service) from Sun [2000b], and ADSI (Active Directory Services Interface) from Microsoft [2000b].

The last two groups of potential UMS clients depicted in Figure 7-2 comprise Analysis and Visualization tools. Important research communities like visualization, data mining, and machine learning developed a plethora of methods and tools in the last years (for an overview of academic and commercial resources, we refer to Card et al. [1999], About.com [2000], GMD AiS [2000], KDnuggets [2000], Russell [2000], Kobsa [2001c], and for a discussion of several commercial data mining tools to Woods and Kyral [1997]). Although many of these tools have been originally developed outside of the user modeling community, many of them have been more recently applied to user modeling tasks as well (e.g., segmentation of a customer population using clustering algorithms, analysis of customers' shopping behavior from past purchase data by learning decision rules, learning

[76] More precisely, Deep Map's agent-based communication framework shows minor extensions to, and omissions from, the corresponding FIPA standards for a variety of reasons including performance and ease of programming (we refer to EML [1999; 2000] for a more detailed discussion).

relevant characteristics of users that typically respond to particular advertisements by training neural networks [Woods and Kyral, 1997]). It is worthwhile to note that these tools are not limited to model analysis and inspection, but allow for assessing the adequacy of the user modeling techniques employed in the UMS as well. This can be accomplished, for example, by applying competing or alternative methods from data mining products to user model contents and evaluate their results regarding key dimensions like performance and accuracy (cf. Herlocker et al. [1999]).

In the middle of Figure 7-2, we find three sub-systems of the Directory Component:

- Communication allows UMS clients to access the UMS via FIPA^{DM} [77], LDAP, and ODBC (for more information on these modules, we refer to Chapter 8.1):
    - The FIPA^{DM} interface module mediates between Deep Map's messaging framework and the functionally structured LDAP interface of the UMS.
    - The LDAP communication module is provided by Directory Server.
    - A commercially available ODBC/LDAP gateway accepts ODBC requests and transforms them into standard LDAP operations and vice versa.
- The Representation sub-system hosts models about (i) relevant characteristics of individual users, (ii) usage of user-adaptive applications, (iii) a domain taxonomy, and (iv) various configuration information. For more information on these models, we refer to Chapter 8.2. The Representation sub-system of the UMS is provided by Directory Server from iPlanet.
- The main task of the Scheduler is to mediate between the Directory Component and the User Modeling Components depicted on the right side of Figure 7-2. A second task of the Scheduler is the provision of LDAP-compliant user modeling functionality (e.g., for creating and deleting user models). For more information on this sub-system, we refer to Chapter 8.3.

On the right side of Figure 7-2, we find those User Modeling Components that are currently implemented in the UMS for Deep Map (for more information on these components, we refer to Chapters 8.4, 8.5, and 8.6):

- The User Learning component (abbreviated 'ULC') learns user interests and preferences from usage data and updates individual user models with associated probabilities (e.g., a user has a presumed interest in churches with a probability of 0.8).
- The Mentor Learning component (abbreviated 'MLC') predicts missing values in individual user models from models of a set of similar users.
- The Domain Inferences component (abbreviated 'DIC') infers interests and preferences in individual user models by applying domain inferences to assumptions that were explicitly provided by users and implicitly acquired by the ULC and the MLC.

Techniques employed in these components include univariate significance analysis (cf. Mitchell [1997], Pohl et al. [1999], and Schwab and Pohl [1999]), memory-based Spearman correlation, various weighted prediction algorithms from the area of

[77] We use the term FIPA^{DM} (i.e., FIPA Deep Map) in the following for referring to Deep Map's limited degree of FIPA compliance (see Chapter 8.1.1 for a brief discussion).

collaborative filtering (cf. Herlocker et al. [1999]), and domain inference rules that take advantage of the domain taxonomy (cf. Kobsa et al. [1994]). These methods seem to be appropriate for the mainly content-based learning tasks we identified in WebGuide. Their tuning, extension, and even replacement may become necessary, however, as soon as user modeling needs evolve (e.g., in case of additional Deep Map components adding to the range of user modeling tasks for the UMS).

8 User Modeling Server for Deep Map: Components

In the following sub-chapters, we present and discuss each of the aforementioned subsystems and components in greater detail.

8.1 Communication

8.1.1 FIPA^{DM} Interface

From a software engineering point of view, it seems advisable for a long-term research project like Deep Map to adhere to a software design paradigm that allows for maximum flexibility. In this vein, agent-based software design seems to be the paradigm of choice. Designing autonomous software agents that communicate via high-level messages significantly reduces coupling between different modules (as opposed to a rather tight coupling between modules when employing established component frameworks like COM) and therefore allows for evolutionary software development. In accordance with FIPA and earlier knowledge sharing initiatives and standards around KQML, Deep Map messages are structured into the three layers *communication*, *messaging*, and *content*. A major difference between many FIPA/KQML implementations and Deep Map is the fact that messages in Deep Map are encoded as Java objects. Since all Deep Map agents are written in Java or, like the FIPA^{DM} interface for the UMS, are encapsulated with a Java wrapper, Java's built-in object serialization capabilities are employed for communicating messages between the different agents. This greatly eliminates the need for parsing messages and improves at the same time the overall performance of the system. The transport of messages is established via the messaging framework JAMFrame [Chandrasekhara, 1999], which is in turn based on the object request broker 'Voyager' from ObjectSpace [2000]. For more information on Deep Map's communication infrastructure, we refer to EML [1999; 2000].

Based on this, the main aim of the FIPA^{DM} interface is to mediate between Deep Map's message-oriented communication framework and the functionally structured LDAP interface of the UMS. Messages sent from Deep Map modules to the UMS are transformed into one or more calls to the LDAP interface of the UMS and vice versa. An example is a query for a user's interest in churches, which is accomplished by calling the search method of the LDAP class LDAPConnection in Java (see also Wilcox [1999]). Proactive communication can also be established by the UMS (e.g., the UMS alerts WebGuide that a user's interest in buildings has significantly increased).

8.1.2 LDAP Interface

Native LDAP connectivity is directly provided by iPlanet's Directory Server (abbreviated 'DS' in the following). DS supports both major revisions of the LDAP protocol, LDAP version 2 and 3. For an overview and a discussion of related RFCs, we refer to Chapters 4 and 5.

In order to ease model management (e.g., the creation and deletion of user models), several extensions to the LDAP protocol have been implemented using LDAP's extended operation facility. The rationale behind this was to relieve administrators and applications from laborious and error-prone administration tasks (e.g., create the initial topography for new user models, set appropriate access control rights, and populate new user models with default assumptions). Another motivation for these user modeling extensions was the preservation of model consistency. New user models, e.g., should become correctly created and initialized, even in case of system breakdowns.

8.1.3 ODBC Interface

ODBC access to the UMS is facilitated via the commercially available product 'PSEnList' from Persistent [2000]. Its interface allows to (i) define relational mappings for the hierarchical LDAP representation used within the UMS (see Chapter 8.2), (ii) access these mappings from a variety of applications via ODBC, and (iii) periodically monitor user model content for certain conditions (e.g., creation of new user models, change of user model content). Appropriate actions can be invoked in such a case (e.g., a re-computation of user group models in a background process when user model contents change).

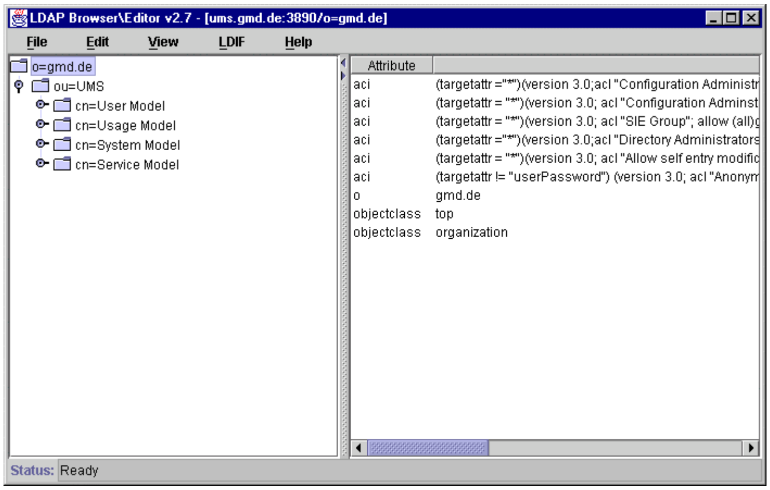
Due to the wide support for ODBC on Windows-based software platforms, the ODBC interface enables many desktop applications (e.g., Microsoft Excel) and a variety of data analysis and visualization tools (e.g., 'sphinxVision' from ASOC [2000], 'InfoZoom' from humanIT [2001]) to directly access the models hosted by the UMS.

8.2 Representation

In the following sub-chapters, we present several examples of the most important models the UMS for Deep Map maintains in its Representation sub-system. We present several screen shots from an LDAP editor/browser [Gawor, 1999], which accesses a development version of the UMS. In each of the following scenarios, we initially identified and authenticated ourselves as an administrator to the UMS. In accordance with the (weak) security and privacy policy currently implemented in this development server, this enabled us to access all test models and retrieve all information contained therein. Presenting the real-world models that are hosted by the UMS at EML's premises would have been a far more intriguing alternative. User privacy, however, required us to restrict ourselves to the development server running at GMD's premises. Please note that the models presented in the following have been developed for the first prototype of WebGuide. Extensions and modifications of the current modeling needs can be anticipated with additional Deep Map components adding to the range of user modeling tasks for the UMS.

The formal definition of the models hosted by the UMS is based on standard LDAP object class and attribute definitions. Nearly all schema elements used in the Representation component are part of the standard LDAP protocol, which is in turn based on the X.500 standard. When modeling the schema for the UMS, we tried to adhere as much as possible to standard schema elements in order to facilitate the deployment of the UMS to other user modeling scenarios.

The current version of the UMS for Deep Map hosts a User Model, a Usage Model, a System Model, and a Service Model. These models can be seen in the left frame of the browser screen shot presented in Figure 8-1.



o=gmd.de
ou=UMS
cn=User Model
cn=Usage Model
cn=System Model
cn=Service Model

Attribute	
aci	(targetattr ="*")(version 3.0;acl "Configuration Administr
aci	(targetattr = "*")(version 3.0; acl "Configuration Adminst
aci	(targetattr = "*")(version 3.0; acl "SIE Group"; allow (all)g
aci	(targetattr ="*")(version 3.0;acl "Directory Administrators
aci	(targetattr = "*")(version 3.0; acl "Allow self entry modific
aci	(targetattr != "userPassword") (version 3.0; acl "Anonym
o	gmd.de
objectclass	top
objectclass	organization

Figure 8-1: User Modeling Server models overview (user attributes only)

In the right frame of the browser, we see various attributes and associated values for the currently selected root entry `gmd.de`. The most prominent attribute `aci` (i.e., access control information) controls access to the models hosted by the UMS. The last value of this multi-valued attribute defines for example that all entries beneath and including `gmd.de` can be accessed by anonymous users (e.g., users that access the UMS anonymously via a Web browser). By default, anonymous users are allowed to perform the LDAP operations read, search, and compare on all entries that are hosted by the UMS[78]. The full access control information that implements this (weak) policy is not entirely visible in Figure 8-1 and looks as follows:

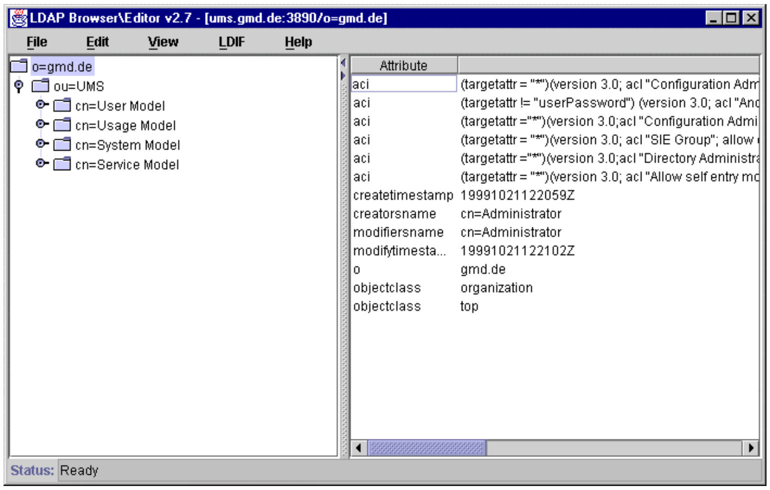
```
 (targetattr != “userPassword”)
 (version 3.0; acl “Anonymous access”;
allow (read, search, compare)
userdn = ”ldap:///anyone”;)
```

Other attributes currently visible in the right frame are the relative distinguished name (i.e. `gmd.de`) and the standard LDAP object classes that are associated with the root entry `gmd.de` (i.e., `top` and `organization`). Each of these object classes defines a list of attributes that are either required or allowed. The list of applicable attributes for an entry like `gmd.de` is defined by the list of its object classes, where each object class adds its attributes to the overall list. Object class `top` is the root class and allows for additional object classes to be associated with `gmd.de`. Object class `organization` adds several required and optional attributes including organization name, postal address, and telephone number. Only those attributes for `gmd.de` that actually contain one or more values are

[78] Such a weak access control policy (i.e., anonymous users are allowed to access all entries that are hosted by the UMS) seems only adequate for a development server that contains no real user data.

shown in Figure 8-1. For more information on LDAP's Information Model and Naming Model, we refer to Chapters 5.1 and 5.2.

Those attributes of `gmd.de` we presented so far are so-called *user attributes*, i.e. attributes that may be modified by clients of the UMS, presupposed that they have appropriate access permissions. Figure 8-2 depicts all attributes of the root entry `gmd.de`, including the *operational attributes* `createtimestamp`, `creatorsname`, `modifiersname`, and `modifytimestamp`.



o=gmd.de
ou=UMS
cn=User Model
cn=Usage Model
cn=System Model
cn=Service Model

Attribute	
aci	(targetattr = "*")(version 3.0; acl "Configuration Adm
aci	(targetattr != "userPassword") (version 3.0; acl "And
aci	(targetattr ="*")(version 3.0;acl "Configuration Admi
aci	(targetattr = "*")(version 3.0; acl "SIE Group"; allow
aci	(targetattr ="*")(version 3.0;acl "Directory Administra
aci	(targetattr = "*")(version 3.0; acl "Allow self entry mo
createtimestamp	19991021122059Z
creatorsname	cn=Administrator
modifiersname	cn=Administrator
modifytimesta...	19991021122102Z
o	gmd.de
objectclass	organization
objectclass	top

Figure 8-2: User Modeling Server models overview (all attributes)

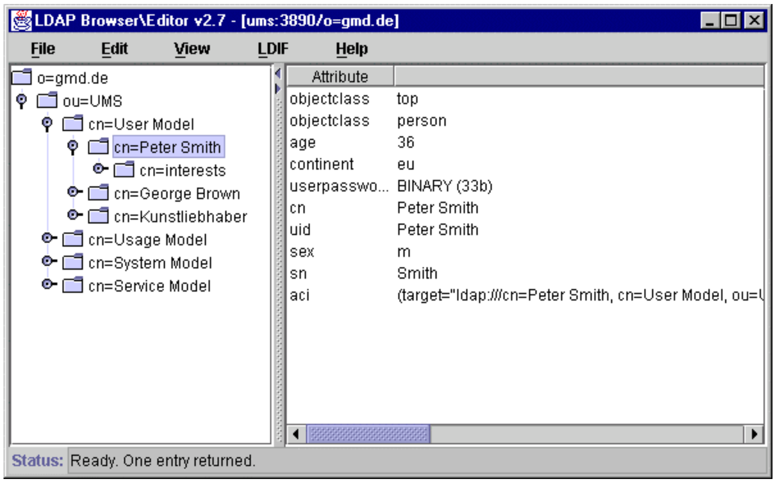
DS updates these attributes each time an entry is modified. Creators and modifiers of an entry include 'real' users, client programs, and also User Modeling Components of the UMS. Operational attributes are a valuable and reliable[79] source of administrative information about model entries. The UMS exploits the information contained in operational attributes for various purposes, e.g. for prioritizing assumptions about a user's interest depending on their origin (similar prioritization schemes can be found, e.g., in the user modeling shell system BGP-MS [Kobsa and Pohl, 1995]). Operational attributes are normally not returned to clients of the UMS, unless clients explicitly request for them.

In the following sub-chapters, we briefly describe each of the models introduced so far. Unless explicitly mentioned, we restrict ourselves to the presentation and discussion of user attributes for model entries.

[79] Neither ordinary users nor administrators are able to modify operational attributes.

8.2.1 User Model

Figure 8-3 depicts in the left frame three user models, one for Peter Smith, one for George Brown, and one for a stereotype called Kunstliebhaber (i.e., art lover). In general, user models comprise a demographic part, which is mainly based on standard LDAP object class and attribute definitions, and a part for users' interests and preferences. The demographic attributes for Peter Smith (his entry is currently selected in the left frame) are shown in the right frame of Figure 8-3.



o=gmd.de
ou=UMS
cn=User Model
cn=Peter Smith
cn=interests
cn=George Brown
cn=Kunstliebhaber
cn=Usage Model
cn=System Model
cn=Service Model

Attribute	
objectclass	top
objectclass	person
age	36
continent	eu
userpasswo...	BINARY (33b)
cn	Peter Smith
uid	Peter Smith
sex	m
sn	Smith
aci	(target="ldap:///cn=Peter Smith, cn=User Model, ou=U

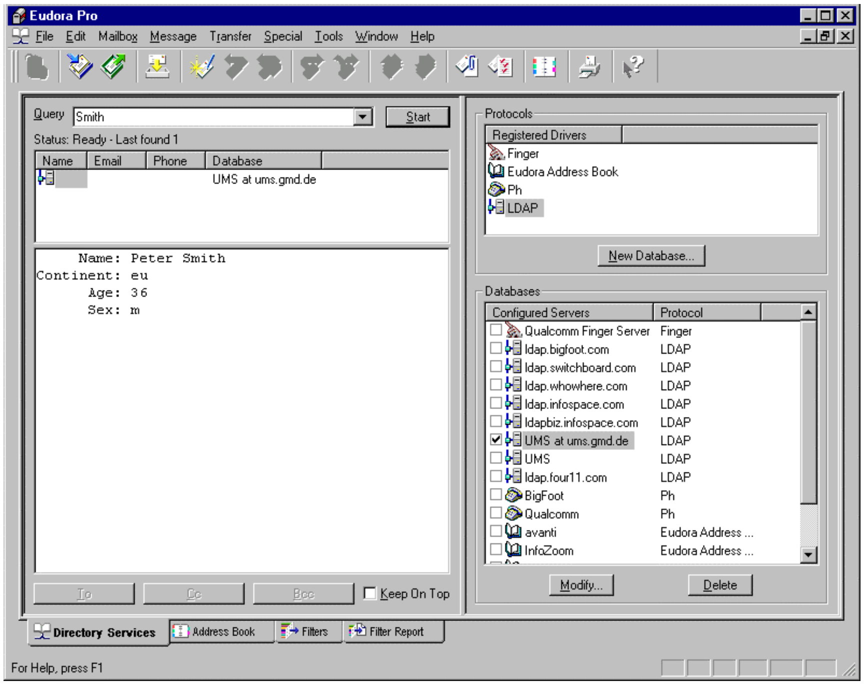
Status: Ready. One entry returned.

Figure 8-3: User models

Based on the assignment of the entry Peter Smith to the object classes top and person, the demographic part comprises required attributes (i.e., common name and surname, abbreviated cn and sn) and optional attributes (e.g. userpassword[80]). And as already mentioned for the object class organization, some more attributes (e.g., description, telephone number) are inherited from the object class person, but not filled with values for Peter Smith yet. Other visible attributes that have been added to meet the specific information needs of WebGuide include age, continent, and sex.

So far, we used an LDAP editor/browser for inspecting the models hosted by the UMS. In Figure 8-4 we contrast this with an interface that facilitates searching model contents. The interface shown there is a part of the e-mail client application 'Eudora' from Qualcomm [2000]. After launching Eudora, we created an LDAP data source called UMS at ums.gmd.de and selected this source for subsequent queries. This is indicated in the bottommost frame on the right by the cross in front of the line UMS at ums.gmd.de. Subsequently, we entered the string Smith into the field named Query on the left and pressed the button labeled Start. The frame below the query field indicates that one record matches this query. Its contents is shown in the frame below. The number of attributes, their naming within the interface, and the visualization of attribute values can be widely adapted by the user and does not require any programming efforts.

[80] The user password shown here is an encrypted representation of the actual one.



Query: Smith

Status: Ready - Last found 1

Name	Email	Phone	Database
			UMS at ums.gmd.de

```
     Name: Peter Smith
Continent: eu
      Age: 36
      Sex: m
```

☐ Keep On Top

Protocols

Registered Drivers
Finger
Eudora Address Book
Ph
LDAP

Databases

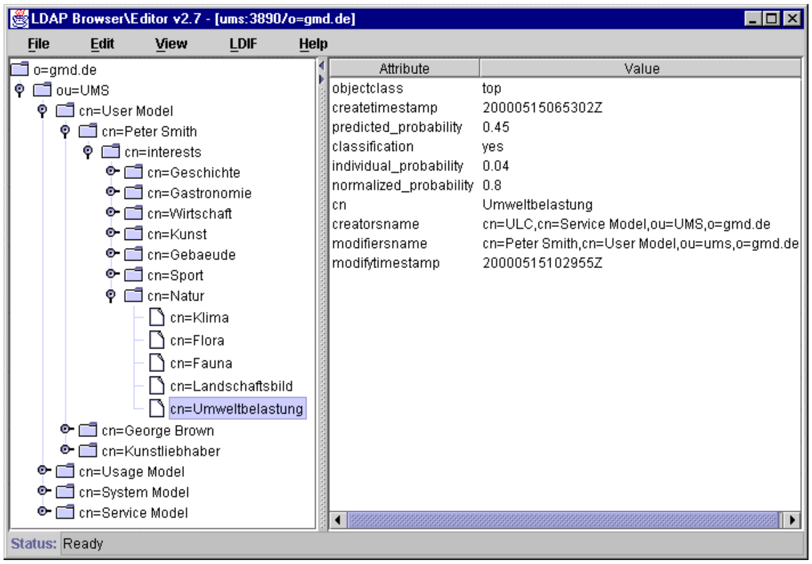
Configured Servers	Protocol
☐ Qualcomm Finger Server	Finger
☐ ldap.bigfoot.com	LDAP
☐ ldap.switchboard.com	LDAP
☐ ldap.whowhere.com	LDAP
☐ ldap.infospace.com	LDAP
☐ ldapbiz.infospace.com	LDAP
☑ UMS at ums.gmd.de	LDAP
☐ UMS	LDAP
☐ ldap.four11.com	LDAP
☐ BigFoot	Ph
☐ Qualcomm	Ph
☐ avanti	Eudora Address ...
☐ InfoZoom	Eudora Address ...

Figure 8-4: User model query for Smith

The major portion of a user model is devoted to users' interests and preferences. The topography and terminology of this part corresponds to the domain taxonomy of Deep Map[81], which is maintained in the System Model (see Chapter 8.2.3). The reference language used in the domain taxonomy for Deep Map is German. This is the reason why all identifiers for interests and preferences used within the UMS for Deep Map are in German.

Within a user model, the top entry for the interests part is labeled `interests`. Figure 8-5 depicts the user model of Peter Smith with his `interests` being unfolded in the left frame. Interests can be hierarchically ordered, e.g. `interests` comprises interest in `Geschichte` (i.e. history), `Gastronomie` (i.e. gastronomy), `Wirtschaft` (i.e. economy), `Kunst` (i.e. art), `Gebaeude` (i.e. buildings), `Sport`, and `Natur` (i.e. nature).

[81] In order to cater to deployment scenarios where a domain taxonomy is not completely specified a priori (i.e., an open corpus of terms), this compliance can be weakened or even abandoned. Compliance with the domain taxonomy can be controlled via several configuration parameters (see Chapter 8.4.3).



- o=gmd.de
  - ou=UMS
    - cn=User Model
      - cn=Peter Smith
        - cn=interests
          - cn=Geschichte
          - cn=Gastronomie
          - cn=Wirtschaft
          - cn=Kunst
          - cn=Gebaeude
          - cn=Sport
          - cn=Natur
            - cn=Klima
            - cn=Flora
            - cn=Fauna
            - cn=Landschaftsbild
            - cn=Umweltbelastung
      - cn=George Brown
      - cn=Kunstliebhaber
    - cn=Usage Model
    - cn=System Model
    - cn=Service Model

Attribute	Value
objectclass	top
createtimestamp	20000515065302Z
predicted_probability	0.45
classification	yes
individual_probability	0.04
normalized_probability	0.8
cn	Umweltbelastung
creatorsname	cn=ULC,cn=Service Model,ou=UMS,o=gmd.de
modifiersname	cn=Peter Smith,cn=User Model,ou=ums,o=gmd.de
modifytimestamp	20000515102955Z

Figure 8-5: Interest model of Peter Smith (all attributes)

The interest in `Umweltbelastung` (i.e., environmental burden), which is a sub-entry of `Natur`, is currently selected. User attributes and operational attributes for this entry are shown in the right frame[82]. The most important attributes include the following:

- `Predicted_probability` is a prediction about a user's interest based on his similarity with other users or stereotypes. In our scenario, `Peter Smith` was found to be similar to the stereotype `Kunstliebhaber` (i.e., art lover). This prediction was calculated based on the presumable interest of this user group in `Umweltbelastung`, and the degree of similarity between him and this group (see Chapter 8.5 on the MLC).
- `Classification` indicates whether a user's `normalized_probability` (see below) of interest is significantly high, significantly low, or not significant. `Peter Smith`'s interest in `Umweltbelastung` was considered to be significantly high (as indicated by '`yes`'). This assumption was calculated by the ULC (see Chapter 8.4).
- `Individual_probability` is an assumption about a user's interest based on the types of information she retrieves from the system (see Chapter 8.4 on the ULC). The probability of `Peter Smith`'s interest in `Umweltbelastung` is considered to be quite low (namely 0.04).
- `Normalized_probability` rates an individual user's interest, as indicated by `individual_probability`, in relation to the interests of the whole user population. For `Peter Smith`, this probability is considered to be fairly high (namely 0.8), since most other users are presumed to have a much lower interest in `Umweltbelastung` than him (see Chapter 8.4 on the ULC).
- `Creatorsname` indicates the creator of this entry (namely the User Modeling Component ULC).

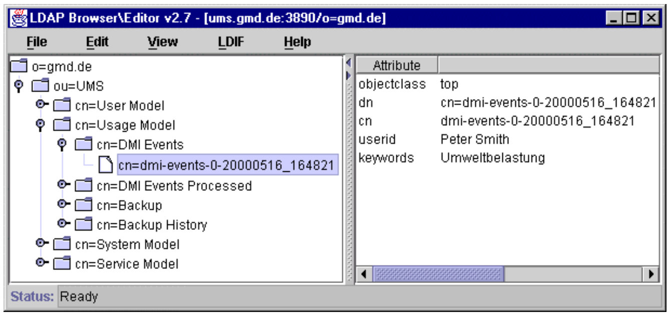
[82] If no specific sorting order is specified by the client, the server returns attributes in an arbitrary order.

- `Createtimestamp` contains the time of creation.
- `Modifiersname` indicates the user who last modified this entry (namely `Peter Smith` himself).
- `Modifytimestamp` contains the time at which the last modification occurred.

The list of attributes for a user's interest we presented is not complete, several more are defined in the schema of the UMS for Deep Map (e.g., an attribute for inferring additional interests by applying domain inferences to assumptions about a user's interests and preferences). For more information on the role of the User Model in the user modeling process, we refer to Chapters 8.4, 8.5, and 8.6.

8.2.2 Usage Model

The Usage Model acts as a persistent storage for usage-related data within the UMS. It comprises usage data communicated by WebGuide, and information related to the processing of these data in User Modeling Components (e.g., a counter for `Peter Smith`'s interface events related to `Umweltbelastung`). Access to the Usage Model is granted to WebGuide and those User Modeling Components that process usage data (e.g., the ULC). Users of WebGuide are not allowed to inspect their usage model. In the left frame of the editor depicted in Figure 8-6, we see the Usage Model from an administrator's point of view.



```
o=gmd.de
  ou=UMS
    cn=User Model
    cn=Usage Model
      cn=DMI Events
        cn=dmi-events-0-20000516_164821
      cn=DMI Events Processed
      cn=Backup
      cn=Backup History
    cn=System Model
    cn=Service Model
```

Attribute	
objectclass	top
dn	cn=dmi-events-0-20000516_164821
cn	dmi-events-0-20000516_164821
userid	Peter Smith
keywords	Umweltbelastung

Status: Ready

Figure 8-6: Usage model

The Usage Model comprises the following parts:

- `DMI Events` contains usage data communicated by WebGuide[83]. Each entry in this sub-tree describes a WebGuide interface event in terms of one or more interests from the domain taxonomy that can be attributed to the user based on this event. For instance, `Peter Smith`'s request for a document about the environmental impacts of tourism is described in terms of an attributed interest called `Umweltbelastung`. The currently selected entry in the left frame and the associated information in the right frame illustrate this.

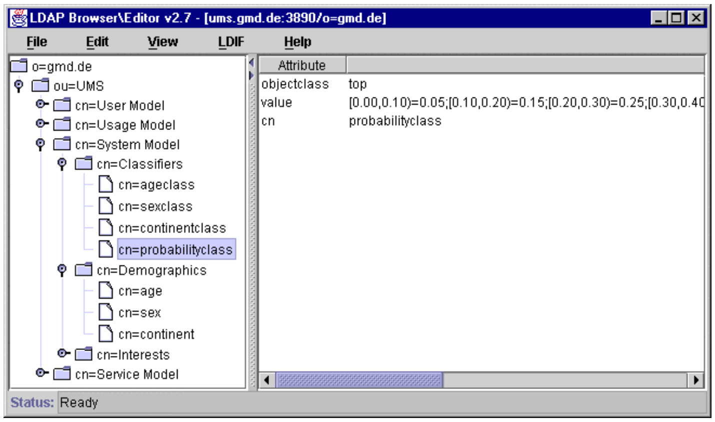
[83] As the label `DMI Events` (i.e., Deep Map Interface Events) for this sub-tree suggests, we anticipate a much broader range of input data for the UMS in the near future.

- `DMI Events Processed` includes information that is required for, and results from, processing usage data contained in `DMI Events` (e.g., the aforementioned event counter for `Umweltbelastung`).
- `Backup` and `Backup History` may contain events from `DMI Events` that have already been processed by User Modeling Components. The main motivation for stockpiling interface events is to preserve them for further processing and analysis, e.g. by employing external visualization and data mining tools (see Figure 7-2).

For more information on the Usage Model and its role in the user modeling process, we refer to Chapter 8.4 that deals with learning about the user.

8.2.3 System Model

The System Model encompasses information about the application domain that is relevant for User Modeling Components of the UMS. Its most important content is the aforementioned domain taxonomy. Access to the System Model is granted to administrators and all User Modeling Components of the UMS (e.g., the MLC). Users of WebGuide are not allowed to inspect the System Model.



```
o=gmd.de
  ou=UMS
    cn=User Model
    cn=Usage Model
    cn=System Model
      cn=Classifiers
        cn=ageclass
        cn=sexclass
        cn=continentclass
        cn=probabilityclass
      cn=Demographics
        cn=age
        cn=sex
        cn=continent
      cn=Interests
    cn=Service Model
```

Attribute	
objectclass	top
value	[0.00,0.10)=0.05;[0.10,0.20)=0.15;[0.20,0.30)=0.25;[0.30,0.40
cn	probabilityclass

Figure 8-7: System model: classifiers and demographics

In the current version of the UMS for Deep Map, the System Model comprises the following parts (see Figure 8-7):

- `Classifiers` contains templates that control the discretization of continuous attribute values (e.g., for discretizing users' age into appropriate groups). The MLC uses these templates for preprocessing attribute values before computing correlations. An example template is shown in Figure 8-7 for the entry `probabilityclass`, which is currently highlighted in the left frame. `probabilityclass` is associated, e.g., to the attribute `Umweltbelastung` in the domain taxonomy (see Figure 8-8). The interpretation of the first part of this template (i.e., `[0.00,0.10)=0.05`) can be outlined as follows: probabilities between 0.00 (inclusive) and 0.10 (exclusive) are associated with the class's mean 0.05. The remainder of the template can be interpreted accordingly.

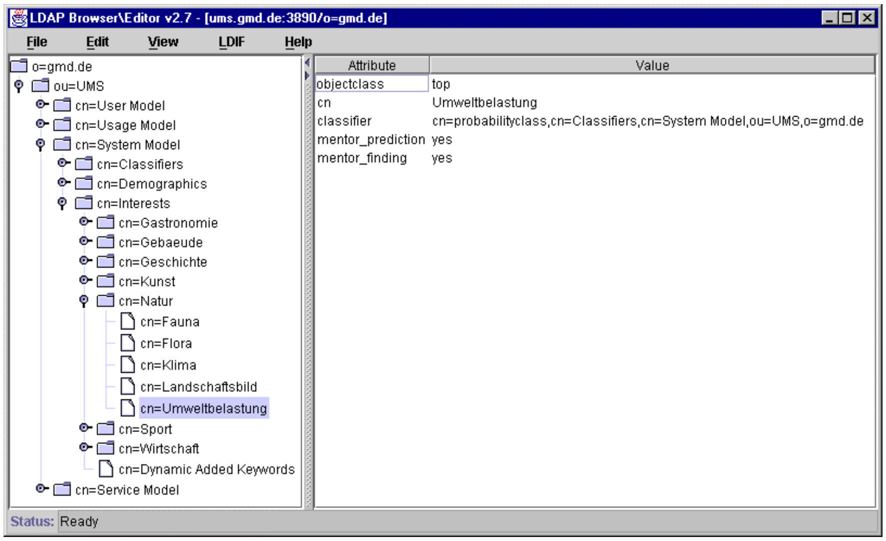
In general, the syntax used for classification templates is as follows (in EBNF, i.e. Extended Backus Naur Form):

```
attribute-classifier ::= classifier {″;″ classifier}*
classifier ::= (range | limit) ″=″ number
range ::= (″(″ | ″[″) limit ″,″ limit (″)″ | ″]″)
limit ::= number | string | ″*″
number ::= {digit}+ [″.″ {digit}*]
string ::= ″″″ {digit | alpha}+ ″″″
```

The creation and management of these templates, however, can be presumed a cumbersome task. Additional support for this task can be provided through a dedicated administration interface and additional syntax checks within the Representation subsystem. In the current version of the UMS, however, such a support is not provided. We recommend to consider this issue for further work on the UMS.

- `Demographics` specifies those attributes (e.g. `age`) in the demographic part of a user model that can be used for finding groups of similar users. In the current implementation of the UMS for Deep Map, this information is mainly relevant for the MLC (see Chapter 8.5). With further components of Deep Map adding to the range of user modeling tasks for the UMS, however, this information might get also relevant for user modeling components that compute models of user groups based on similar characteristics in individual user models, irrespective of the techniques they employ for this task (e.g., Bayesian clustering for acquiring explicit models of user groups [Orwant, 1995; Paliouras et al., 1999])[84].
- `Interests` mirrors the domain taxonomy from the Deep Map database. In its current version, the domain taxonomy covers seven areas of interest (namely restaurants, buildings, history, art, nature, sports, and economy). The `interests` sub-tree comprises five levels with nearly 500 leaf entries. Figure 8-8 shows a small portion of the domain taxonomy. `Umweltbelastung` is currently selected in the left frame and its attributes are shown in the right frame. The attribute value for `classifier` specifies the classification template to be used for discretizing interest probabilities for `Umweltbelastung`. The attributes `mentor_prediction` and `mentor_finding` control whether predictions based on similar users should be computed for this interest and whether this attribute should be included in the mentor finding process. Both flags are on for `Umweltbelastung`.

[84] This is in contrast to the MLC, which predicts missing entries in individual user models from entries in a set of similar user models (i.e., no models of user groups are computed).



```
o=gmd.de
  ou=UMS
    cn=User Model
    cn=Usage Model
    cn=System Model
      cn=Classifiers
      cn=Demographics
      cn=Interests
        cn=Gastronomie
        cn=Gebaeude
        cn=Geschichte
        cn=Kunst
        cn=Natur
          cn=Fauna
          cn=Flora
          cn=Klima
          cn=Landschaftsbild
          cn=Umweltbelastung
        cn=Sport
        cn=Wirtschaft
        cn=Dynamic Added Keywords
    cn=Service Model
```

Attribute	Value
objectclass	top
cn	Umweltbelastung
classifier	cn=probabilityclass,cn=Classifiers,cn=System Model,ou=UMS,o=gmd.de
mentor_prediction	yes
mentor_finding	yes

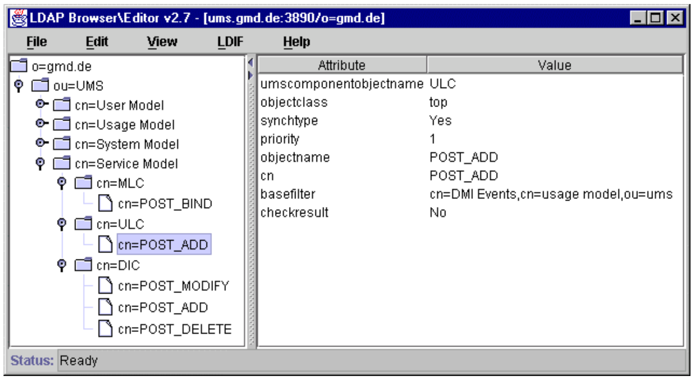
Figure 8-8: System model: domain taxonomy

For more information on the System Model including a usage scenario, we refer to Chapter 8.5.

8.2.4 Service Model

The Service Model contains information that is required for establishing communication between the Directory Component and the User Modeling Components. Access to the Service Model is restricted to administrators and User Modeling Components (e.g., the DIC). Users of WebGuide are not allowed to inspect the Service Model.

The Service Model is divided into three parts, each of them being dedicated to a single User Modeling Component (e.g., the ULC). Each of the entries contained in these parts represents a description of a server-internal event type in which a User Modeling Component is interested. Figure 8-9 depicts an example of such a subscription.



```
o=gmd.de
  ou=UMS
    cn=User Model
    cn=Usage Model
    cn=System Model
    cn=Service Model
      cn=MLC
        cn=POST_BIND
      cn=ULC
        cn=POST_ADD
      cn=DIC
        cn=POST_MODIFY
        cn=POST_ADD
        cn=POST_DELETE
```

Attribute	Value
umscomponentobjectname	ULC
objectclass	top
synchtype	Yes
priority	1
objectname	POST_ADD
cn	POST_ADD
basefilter	cn=DMI Events,cn=usage model,ou=ums
checkresult	No

Figure 8-9: Service model

The entry currently selected in the left frame specifies a subscription for additions to the Usage Model (more precisely, a subscription for notification regarding such additions). The attributes shown in the right frame specify this notification request. The value for attribute umscomponentobjectname refers to the User Modeling Component that induced this request (i.e. the ULC)[85]. The attributes cn and objectname denote the type of LDAP request that should be monitored (namely add operations). Two types of events can be raised for a single LDAP operation, one before and one after an LDAP operation has been executed by the server. The first type is prefixed with PRE, the second with POST. In our example, all additions to the Usage Model are first executed by the server and then communicated to the ULC (as indicated by POST_ADD). Post-notifications allow a User Modeling Component to react on the outcome of an LDAP operation (e.g., start learning a persistent entry, invoke an undo operation). Pre-notifications enable a User Modeling Component to get invoked before an LDAP operation is executed by the server (e.g., carrying out consistency checks, preparing for rollback). In general, the following LDAP operations can be tracked by the Scheduler before and after execution: bind, unbind, search, modify, add, delete, rename, compare, and abandon. User modeling operations that are implemented as extensions to the LDAP protocol can also be monitored and communicated to User Modeling Components (e.g., for creating and deleting user models).

An important attribute of an event subscription is basefilter. Its value allows to restrict the portion of the overall directory tree that is to be monitored. In the example depicted in Figure 8-9, this monitoring space is defined as cn=DMI Events, cn=usage model, ou=UMS, o=gmd.de. Based on this, the Scheduler communicates all additions of entries that take place in the hierarchy below this entry to the ULC. Additional attributes of an event subscription are (i) synchtype, which specifies whether the Scheduler communicates an event in a synchronous or asynchronous manner, (ii) priority, which directly maps to the priority used for communicating events via the ORB, and (iii) checkresult, which specifies whether the Scheduler should check for each event being successfully communicated to a User Modeling Component before resuming processing. For more details on how the Service Model is used by the Scheduler, we refer to Chapter 8.3.

The centralization of configuration information in the Service Model can be assumed to considerably ease the administration of the UMS, as opposed to the efforts necessary for maintaining a plethora of (potentially distributed) configuration files. Further work on the Service Model may additionally take advantage of

i. LDAP's facilities for replicating parts of a directory tree across several servers,
ii. more recent proposals for LDAP standards that aim at representing Java objects (see Ryan et al. [1999a]) and CORBA object references (see Ryan et al. [1999b]) in LDAP representations, and
iii. a recent proposal for LDAP entries with a transient lifetime [Yaacovi et al., 1999].

[85] This name denotes also the service name a component uses for registering to the ORB. At runtime, the Directory Component dynamically binds to User Modeling Components based on their service names.

These facilities would allow for (temporarily) replicating models hosted by the UMS (e.g., user models) and associated user modeling functionality (e.g., the ULC) across a network of computers. Together with today's facility of the UMS for remotely 'plugging' new user modeling components into the server at runtime, this infrastructure provides a promising basis for a new generation of user modeling applications, which supports, e.g., nomadic user models and associated user modeling functionality (cf. Kobsa [2001a]).

8.3 Scheduler

In the following sub-chapters, we first describe the main tasks of the Scheduler. After that, we present an example that demonstrates the processing of event subscriptions from the Service Model and the communication of events to User Modeling Components. Finally, we briefly sketch some implementation details.

8.3.1 Introduction

The main task of the Scheduler is to mediate between the Directory Component and the User Modeling Components. User Modeling Components can subscribe to certain types of UMS events by maintaining event subscriptions in the Service Model (see Chapter 8.2.4). This approach limits the amount of communication, allows for adding and removing user modeling components at runtime, and distributing them dynamically across a network of computers. After the launch of the UMS, the Scheduler loads event subscriptions from the Service Model. Subsequently, the Scheduler periodically checks the Service Model for new entries and, if necessary, updates its internal subscription tables accordingly. Henceforth, the Scheduler acts as a kind of event broker that supervises LDAP events within the UMS and communicates them together with associated data to User Modeling Components. In the current version of the UMS for Deep Map, communication between the Scheduler and User Modeling Components is established via the commercially available ORB 'VisiBroker' from Inprise [2000].

A second task of the Scheduler is the provision of user modeling extensions to the LDAP protocol. If a new user model has to be created, e.g., this comprises the execution of several standard LDAP operations in a particular order, namely (i) checking for an already existing model, (ii) establishing the basic topography of a new model, (iii) setting appropriate access rights, and (iv) populating the model with default values. Moreover, rollback mechanisms have to be provided that preserve model consistency in case of potential problems during the creation process. Centralizing these administration tasks in the Scheduler preserves model consistency and relieves administrators and application programmers from laborious and error-prone administration and programming tasks. In the current version of the UMS for Deep Map, we implemented two operations for creating and deleting a user model using the standard mechanisms for adding custom extensions to the standard LDAP protocol.

8.3.2 Usage Scenario

In the following, we briefly describe a scenario in which the Scheduler communicates the addition of a usage event to the ULC. Figure 8-10 depicts the most important components that are involved in this example, namely the Directory Server, the Scheduler, and the ULC.

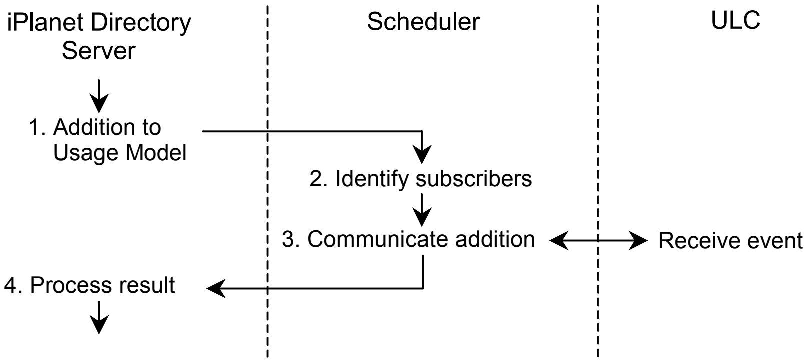


Figure 8-10: Scheduling scenario

The following steps constitute our scenario (the numbers in the following list refer to those in the figure):

1. A user's request for a Web document results in the communication of an event vector to the UMS that includes the term `Umweltbelastung`[86]. This vector is inserted into the `DMI Events` part of the Usage Model using an LDAP add operation (see also Figure 8-6).
2. Subsequently, the add event is handed over to the Scheduler, which starts scanning its internal subscription tables for matching entries[87]. The event type and the basefilter (namely `cn=DMI Events, cn=usage model, ou=UMS, o=gmd.de`) of the ULC subscription depicted in Figure 8-9 match the current event; hence, the Scheduler prepares this event for communication to the ULC (in the case of several matching subscriptions, the event is communicated to all subscribers).
3. The Scheduler communicates the add event and associated data (mainly the event vector) to the ULC, thereby following the processing specification contained in the subscription (as specified by the attributes `synchtype`, `priority`, and `checkresult`). Taking the event filters applied in the previous step and the operational attribute `creatorsname` of a usage event into account, the Scheduler can avoid potential recursion problems that may result from the ULC adding entries to the Usage Model hosted by the UMS.

[86] This term can be regarded as characterizing the content of the requested hypermedia document (there may be more than one descriptive term for a page). It could come from the HTML 'description' and 'keywords' tags, or it could have been selected using a term significance measure such as DF/ITF [Sparck Jones, 1972].

[87] The Scheduler uses a quite similar subscription mechanism for integrating with DS. The Scheduler transforms event subscriptions from User Modeling Components into event subscriptions to DS. This approach successfully minimizes the amount of system resources necessary for event management purposes (see also the results of our empirical experiments in Chapter 9).

4. The Scheduler reports the successful event submission to DS, which resumes processing the LDAP add operation.

8.3.3 Implementation

In order to operate as efficient as possible, the Scheduler is tightly integrated with DS (see Figure 8-11).

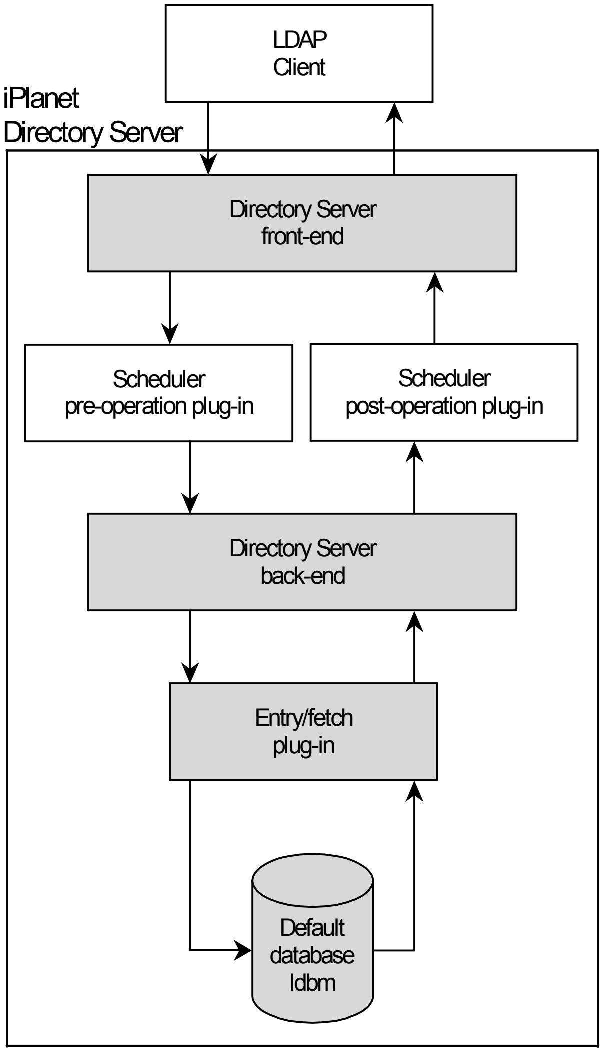


Figure 8-11: Scheduler integrated with Directory Server

On the top, we see an *LDAP Client* accessing the server. Immediately below that, the *Directory Server front-end* is depicted. Its main tasks include processing of incoming LDAP requests, calling the back-end for database access, and sending results back to the client. Below that, the Scheduler is depicted as a *pre-operation* and as a *post-operation plug-in*. Beneath these plug-ins, we see the *Directory Server back-end*, which is mainly responsible for database access. An additional plug-in handles all low-level entry/fetch operations just before the database is accessed.

In the current version of the UMS for Deep Map, we use the standard *Entry/fetch plug-in* of DS, which interfaces to the *default database* management system *ldbm*. A potential line of further development is the replacement of this plug-in with a custom one, thereby enabling access to a relational database management system (e.g., from Oracle). Besides a different performance characteristics (i.e., read performance is traded in for update performance), this would enable the UMS to take advantage of the transactional facilities provided by a database management system.

From an implementation point of view, the Scheduler takes advantage of several dynamic link libraries provided by DS. The programming language used for implementing the Scheduler is C++. Access to the ORB is facilitated via the commercially available product ‘VisiBroker for C++’ from Inprise [2000].

8.4 User Learning

In this sub-chapter, we start with a brief overview of related work on acquiring user interests from usage data. Based on this review and the user modeling requirements we identified in WebGuide, we subsequently discuss the selection of a specific technique for learning user interests. After that, we present an example that demonstrates the acquisition of user interests within the ULC (i.e. User Learning Component). In the last part of this sub-chapter, we describe various design and implementation decisions we have taken against the background of performance and scalability requirements.

8.4.1 Introduction

A currently very active strand of user modeling research is devoted to learning users’ interests from relevant features of objects that users have viewed, rated, put in electronic shopping carts, bought, etc. The resulting assessment is often used for recommending and filtering objects (cf. Oard [1997]). This kind of filtering is called *feature-based filtering* (or sometimes *content-based filtering*)[88], since it is based on object features. For instance, assume that users’ interest in movies is determined by movie features like genre, actors, director, etc. A learning algorithm would then attempt to learn a user’s preferences with respect to these features based on her interaction with movies and related information, and thereafter rate new movies whether they are presumably interesting to this user. The result can be exploited, e.g., for recommending movies that this user will presumably rate highly, and for supplying and emphasizing features about movies that are presumably relevant.

Plenty of work has already been carried out in the area of learning about users’ interests. An early system by Jennings and Higuchi [1993] employed neural networks for filtering Internet news before presenting them to users. Other work includes ‘Fab’ [Balabanovic, 1997; Balabanovic and Shoham, 1997], ‘Letizia’ [Lieberman, 1995], and ‘LaboUr’ [Pohl et al., 1999; Schwab and Pohl, 1999; Schwab et al., 2000]. While these systems used a single technique for a specific learning task, a parallel strand of research evaluated the application of alternative techniques to the same learning tasks (e.g., Pazzani and Bilsus [1997], Breese et al. [1998], Herlocker et al. [1999]). In ‘Syskill & Webert’ [Pazzani and Billsus, 1997], for example, several machine learning techniques (namely the nearest-neighbor algorithm

[88] The term content-based filtering is mainly used in domains where the objects of interest are documents. In this case, the features are those terms of a document that are considered to be representative for the content of the document.

and its PEBLS variant [Cost and Salzberg, 1993], induction of decision trees with algorithms like 'ID3' [Quinlan, 1986], two neural network approaches, and the naive Bayesian classifier [Duda and Hart, 1973]) were evaluated regarding their performance and accuracy in acquiring interest profiles from explicit user ratings on a set of biomedical documents. Based on this evaluation, the naive Bayesian classifier was chosen as the default algorithm for Syskill & Webert. Further improvements regarding learning accuracy have been achieved by emphasizing those features that are particularly relevant for classifying biomedical documents (e.g., by letting the user select and rate the importance of document features for classification).

Many of the algorithms discussed in the literature (e.g., those evaluated for Syskill & Webert) had to be discarded in our project because of their reliance on negative evidences of user interest. Users are meanwhile known for not giving very much feedback on the appropriateness of presented items, particularly not negative feedback (see the discussion in Schwab and Pohl [1999] and Schwab et al. [2000]). Some systems like Letizia [Lieberman, 1995] compensate this by utilizing those options (namely Web links) that the user did not select as negative evidences of user interest. This however seems hardly appropriate for Web-based systems. If a user clicks on some links on a Web page but not on others, this does not imply that the other links are not interesting to her. This is especially true in cases where a Web page does not fit on a computer screen, e.g. due to screen size, screen resolution, or excessive page length. There is evidence in the hypermedia literature that those parts of a hypermedia page that do not fit on a physical screen (and thus require scrolling in order to be seen) are only viewed (and therefore perceived) by a relatively small number of users [Nielsen, 1996]. For all these reasons, we decided to use learning algorithms that solely rely on positive evidences of user interest.

Of those algorithms that have been developed and evaluated in the literature to determine whether a user is interested in specific object features, we selected the univariate significance analysis (cf. Mitchell [1997] and Schwab and Pohl [1999]). The main reason for this choice is the ability of this algorithm to

i. learn incrementally, which allows for keeping up with users' changing interests and preferences,

ii. represent learning results explicitly, which allows for leveraging synergistic effects between different learning techniques,

iii. employ a domain taxonomy, which can considerably improve the learning process (both in terms of computational complexity and the quality of the learning results),

iv. rely on positive learning examples only, which relieves WebGuide from being forced to collect negative evidences of user interest, and

v. demonstrate scalability, e.g. when the number of users, system's usage, and document features increase.

Univariate significance analysis is a statistical technique that is based on the assumption that the occurrence of an object feature (e.g. `Umweltbelastung`) in users' usage histories is normally distributed (see Figure 8-12)[89]. If a feature appears in an individual user's usage history less frequently than in a random sample, the user can be considered not to be interested in it. If a feature appears more frequently than in a random sample, the user can be assumed to be interested in this feature.

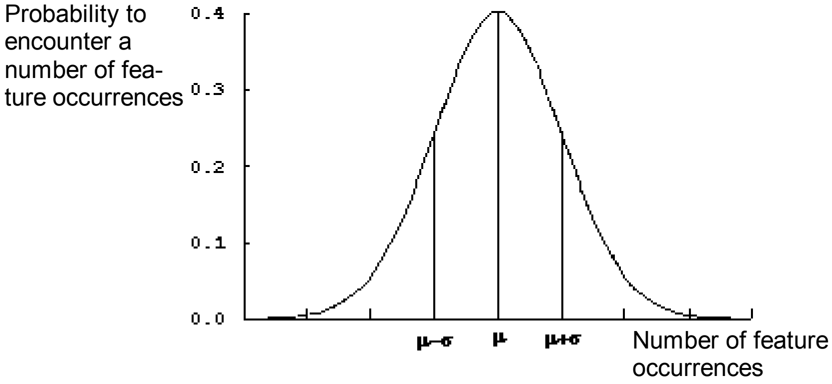


Figure 8-12: Normal distribution of users' interest in an object feature

In order to determine those non-interests and interests that are statistically significant, we introduce two confidence limits c_l and c_u for the lower and upper limit, respectively. If the actual number of feature occurrences in a user's navigation history is below c_l, we classify her as not being interested. If the value is between the two limits, then no assumption is justified. If the number of occurrences is above c_u, we classify her as being interested in an object feature. The three cases are depicted in Figure 8-13.

In conclusion, ULC maintains the following assumptions about a user's interest in an object feature:

- the probability of the user's interest based on feature occurrences in her usage history (henceforth called *individual probability p_i*),
- the probability of the user's interest based on feature occurrences in her usage history in relation to the occurrences distribution for all users (henceforth called *normalized probability p_n*), and
- a *classification* whether the user can be assumed to be interested in an object feature, whether we can assume the user as not being interested, or whether an assumption is not justified.

[89] For future versions of the ULC, we aim at investigating and evaluating alternative statistical distributions. The 'Beta distribution' seems to be especially promising, since it allows for approximating a variety of probability distributions (e.g., linear, normal, exponential, parabola) with a single stochastic model and a few model parameters. An early system that employed the Beta distribution for modeling user interests was the 'Doppelgänger' user modeling server [Orwant, 1995]. In marketing research, the Beta distribution is employed as a stochastic model for quite similar purposes, e.g. for modeling consumer behavior [Lilien et al., 1992].

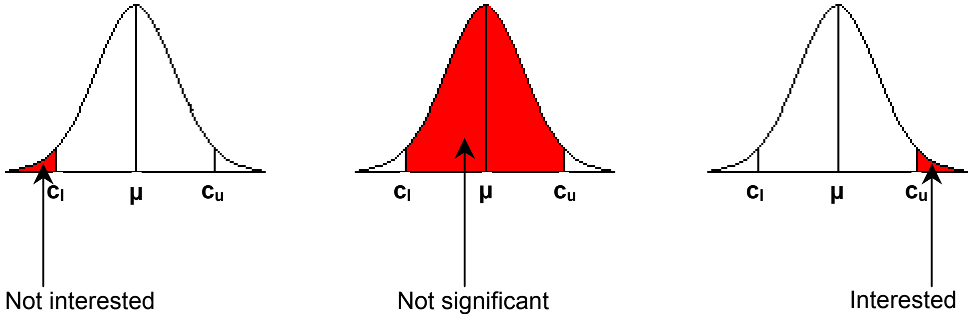


Figure 8-13: Classification of a user's interest

In the following sub-chapter, we introduce a scenario from WebGuide and describe in more detail how the ULC acquires the aforementioned probabilities and the interest classification.

8.4.2 Usage Scenario

In the following scenario, we want to determine whether and to what extent Peter Smith is interested in the feature `Umweltbelastung`. WebGuide sends event vectors containing this feature to the UMS, e.g. whenever a user requests documents that discuss environmental burden in Heidelberg (for an example of such an event vector, we refer to Figure 8-6). We further assume that the UMS already collected in its Usage Model $n = 216$ occurrences of `Umweltbelastung` for all users, and a total of $N = 715$ occurrences of all features for all users. Thus, the probability p to randomly select this feature is[90]

$$p = \frac{n}{N} = \frac{216}{715} = 0.3$$

Formula 8-1: Probability for 'Umweltbelastung' over all features and users

We further assume that WebGuide reported $N_a = 30$ occurrences of all features for Peter Smith, $n_a = 15$ of them for the feature `Umweltbelastung`; hence, the individual probability p_i for Peter Smith is

$$p_i = \frac{n_a}{N_a} = \frac{15}{30} = 0.5$$

Formula 8-2: Individual probability for 'Umweltbelastung'

[90] For bootstrapping purposes, we start with a very small p.

Based on this, the ULC calculates the normalized probability p_n for Peter Smith as follows:

$$p_n = \frac{1}{1+e^{-0.4\cdot\frac{n_a - p\cdot N_a}{\sqrt{p\cdot(1-p)\cdot N_a}}}} = \frac{1}{1+e^{-0.4\cdot\frac{15-0.3\cdot 30}{\sqrt{0.3\cdot 0.7\cdot 30}}}} = 0.72$$

Formula 8-3: Normalized probability for 'Umweltbelastung'

In order to determine whether or not a user's interest is statistically significant, we introduce two confidence limits c_l and c_u:

$$c_{l/u} = \mu \mp z\cdot\sqrt{p\cdot(1-p)\cdot N_a}$$

Formula 8-4: Confidence limits

μ is the means of the distribution and equals to the overall probability p multiplied by the total number of users' event vectors ($p\cdot N_a$), while z is the critical value. It determines the area under the standard normal distribution that falls within the confidence interval. For a confidence rate of 95%, the critical value z is 1.9604. This means that 95% of random samples fall within this interval and 5% lie outside. In order to increase confidence, we have to increase z accordingly (e.g., for a confidence of 99%, z is 2.5762).

The ULC calculates the 95% confidence limits for classifying Peter Smith's interest in `Umweltbelastung` as follows:

$$c_l = 0.3\cdot 30 - 1.9604\cdot\sqrt{0.3\cdot 0.7\cdot 30} = 4.07$$

$$c_u = 0.3\cdot 30 + 1.9604\cdot\sqrt{0.3\cdot 0.7\cdot 30} = 13.92$$

Formula 8-5: Peter Smith's confidence limits for 'Umweltbelastung'

After rounding, c_l and c_u can be interpreted with respect to Peter Smith's interest in `Umweltbelastung` as follows:

- If there are 4 or fewer events with this feature, he can be considered to have no interest in it.
- If the number of events is 14 or above (this is the case in our example with n_a = 15), he can be considered to be interested in this feature.
- Otherwise, we cannot assume a significant interest or non-interest of him.

Concluding, the ULC assesses Peter Smith's interest in Umweltbelastung as follows (see also Figure 8-5, where different probability values are shown):

- an individual probability of 0.5,
- a normalized probability of 0.72, and
- a classification that he can be assumed to be interested in this feature.

After computing these probabilities and classifying Peter Smith's interest, the ULC checks whether his user model already contains an entry for Umweltbelastung and whether the attributes individual_probability, normalized_probability, and classification are present. If the entry Umweltbelastung exists, the ULC checks whether Peter Smith (or an administrator on behalf of him) recently inserted or modified this entry, as reported by the attributes creatorsname and modifiersname (see Chapter 8.2.1). Based on the outcome of these checks, the ULC proceeds as follows:

- If there is no entry for Umweltbelastung, the ULC inserts it. If necessary, the ULC also inserts superordinate entries according to the domain taxonomy (e.g., the more general interest in Natur, see Figure 8-8).
- If Peter Smith (or an administrator on behalf of him) recently inserted or modified assumptions about his interest in Umweltbelastung, an update of these assumptions is not performed. The rationale behind this is that the ULC gives higher priority to user-initiated modifications than to system-initiated ones (a similar prioritization can be found, e.g. in Kay [1995] and Kobsa and Pohl [1995]). Trust in a source is thereby given priority over recency of information when resolving the conflict between an existing user model entry and a newly acquired assumption[91].
- If an entry about the user's interest in Umweltbelastung is already present in the user model and does not strongly differ from the new system-generated value, the update (which is relatively costly in terms of system resources) is also not performed[92].

8.4.3 Implementation

The design and implementation of the User Modeling Components follows established software design principles like transparency, flexibility, and scalability (see also Chapter 6.1 and Tanenbaum [1992]). In order to illustrate this, we briefly present selected design and implementation decisions in this sub-chapter.

[91] See Fink [1999] for a discussion of related consistency problems and their impact on the overall usability of user-adaptive applications.

[92] In general, an update seems dispensable in user modeling scenarios that assign the performance of the user modeling system a higher priority than the recency and accuracy of the user-related information maintained by these systems.

Transparency (of service provision) is achieved by adhering as much as possible to established standards: communication with the Scheduler is accomplished via an LDAP-based CORBA interface and communication with the Directory Component is established via standard LDAP access facilities. CORBA and LDAP offer a variety of advantages, the most notable one is *location transparency* (i.e., communicating components do not rely on location information that is available a priori); hence, components can be flexibly distributed across a network of computers, even at runtime[93]. For relevant communication details (e.g., server names, passwords, names of CORBA services, models' location), the ULC relies on the Service Model (see Chapter 8.2.4) and on a local configuration file. Based on the information therein, the ULC dynamically locates the Scheduler and binds to DS as an ordinary user. In a subsequent step, the event registrations in the Service Model (e.g. POST_ADD) are checked as to whether they comply with current monitoring needs (see Chapter 8.2.4).

Flexibility is accomplished by putting an extensive list of configuration parameters at the disposal of system designers and administrators. These parameters control for example the

- synchronization with the Directory Component (e.g., host name, port number, delay and synchronization interval in case of DS and Scheduler running on the same computer),
- login into the Directory Component (e.g., user name, password),
- location of relevant (sub-) models hosted by the UMS (e.g., User Model, Usage Model, domain taxonomy within the System Model, and Service Model),
- univariate significance analysis (e.g., a timer interval and a minimum number of event vectors for starting the computation, internal precision used for computing and storing probabilities, size of the confidence interval),
- synchronization of the ULC-internal cache with models hosted by the UMS (e.g., handling of updates in the domain taxonomy, handling of features in event vectors that are not contained in the domain taxonomy, attribute names to be used in the User Model for individual and normalized probabilities and the classification, replication of more general interests from the domain taxonomy, forced update of unchanged interests, prioritization of user-modified assumptions over system-modified ones),
- system's logging (e.g., log file device, different levels of logging granularity), and
- backup and recovery details (e.g., deletion of processed event vectors, creation of one or more backups of processed event vectors in a dedicated part of the Usage Model). See Chapter 8.2.2 for more information on this topic.

Configuration parameters allow system designers and administrators to (i) keep pace with minor changes in user modeling needs, (ii) employ differently configured versions of the ULC within the same deployment scenario, and (iii) facilitate the deployment of learning components to new user modeling scenarios. In future versions of the ULC, we foresee to migrate most of these parameters from the local configuration file to the Service Model. This can be expected to further ease the administration of the UMS, since configuration information can then be remotely administered with commonly available software tools (e.g., LDAP browsers). We also aim at reflecting changes in service descriptions from the

[93] Based on that, further work on the UMS may aim at automatically distributing user modeling components across a network of computers according to resource requirements.

Service Model to the ULC at runtime; hence, shutting down parts of the system for activating a new configuration can then be avoided in most cases.

Scalability of the ULC is supported by a variety of design decisions including the following ones:

- The amount of synchronous communication between the ULC, the Directory Component, and other User Modeling Components is reduced to a minimum (see the event subscription mechanism we described in Chapter 8.2.4).
- All relevant models hosted by the UMS (e.g., User and Usage Models, Service Model) are replicated in an internal cache.
- Event vectors submitted by the Scheduler are concurrently managed in a separate queue before being periodically processed.
- Univariate significance analysis is computed only for those users that are currently active. Likewise, model caching is restricted to active users only. This approach does not only increase scalability, but also contributes to the consistency of the user model from a user's point of view (i.e., assumptions about a user's interests do not change without the user actually interacting with the system).
- User model update can be restricted to those cases, where recently acquired interests differ from the ones stored in the User Model.

Against this background, we believe that the processor requirements of the ULC are moderate. And regarding scalability, we assume that its response times linearly grow with the number of user profiles and the workload in small and medium-sized real-world environments. For related feedback from our performance experiments, we refer to Chapter 9.3.

In the current version of the UMS for Deep Map, the ULC runs as a system service on Windows NT and Sun Solaris[94]. The implementation language used is Java 1.2. The communication with the Scheduler is established via the commercial ORB VisiBroker for Java [Inprise 2000]. Access to DS is facilitated via standard LDAP access facilities (i.e., Directory SDK from Netscape [2000b]).

8.5 Mentor Learning

A different approach to the prediction of unknown characteristics of the current user (particularly her interests and preferences) is to compare her with similar users (so-called 'mentors')[95]. In research and commercial environments, this approach is mostly called 'collaborative filtering' [Goldberg et al., 1992] and more recently also 'clique-based filtering'[96] [Alspector et al., 1997; Kobsa et al., 2001].

In the following, we motivate our selection of collaborative filtering algorithms by the user modeling requirements that we elicited for Deep Map. After that, we exemplify the learning

[94] The ULC can be easily deployed to other platforms.

[95] We stick to the term 'mentor' in the remainder of this paper.

[96] The main argument for the renaming is that the term 'collaborative filtering', which was originally coined by Goldberg et al. [1992], is hardly appropriate anymore, mainly due to the emphasis of more recent systems on implicit acquisition techniques and their limited or non-existent support for user collaboration.

approach taken in the MLC (Mentor Learning Component) based on a hypothetical scenario from WebGuide. Finally, we briefly describe selected implementation details.

8.5.1 Introduction

The employment of collaborative filtering in our UMS compensates for some shortcomings of the approach pursued in the ULC, namely frequency analysis and classification of features occurrence (see also Herlocker et al. [1999], Billsus and Pazzani [2000], and Kobsa et al. [2001]):

- The user has to interact with the user-adaptive system for a while before the ULC can determine user interests based on feature occurrence. This may be a critical restriction, e.g. in applications where a user model is maintained during a single session only and in scenarios where users expect immediate personalized services in return for, e.g. granting permission that their personal data may be stored and processed for adaptation purposes.
- Content-based features may be not the main determinants for users' interest in objects. This is particularly true when users' personal tastes, esthetical judgements, etc. come into play.
- Arbitrary restrictions (e.g., limited bandwidth and online fees in case of a modem connection) or simply unawareness may prevent a user from requesting interesting objects.

And, vice versa, core problems of collaborative filtering (e.g., sparse population of the matrix of users and content features, startup problems in scenarios where new objects and features are frequently added to the system) that are reported in the literature [Soboroff et al., 1999; Billsus and Pazzani, 2000; Kobsa et al., 2001] can be partially alleviated in our architecture by

i. interests and preferences acquired by the ULC,
ii. user models that represent stereotypes[97] (i.e., standard assumptions that a system designer makes about members of a category like art lovers [Rich, 1979; 1983; 1989]), and
iii. information explicitly provided by users (e.g., demographic information, probabilities of interest).

Early work on the inclusion of models of similar users into the user modeling process was carried out by, e.g. Orwant [1995]. His system 'Doppelgänger' is a user modeling server that periodically (like every night) applies a clustering algorithm to models of individual users. Clusters found during this process are represented in models of user groups. As is the

[97] The availability of empirical data about tourists' interests in Heidelberg also motivated this decision. Future research may prove, however, whether this approach is viable (i.e., users' interests may deviate according to different task foci when visiting the 'real' and the virtual city of Heidelberg on the Web).

case with stereotype methods (see e.g. Rich [1979; 1983; 1989]), information from group models can henceforth be employed when information in individual models is missing. An important difference to stereotype reasoning is that changes in individual user models can be taken into account by regularly re-applying the clustering procedure. This is not likely to occur often though since clustering is known to be computationally expensive [Fisher, 1987; 1996] and in practice needs to be supervised by experts who analyze the clusters and iteratively refine the clustering process. More recently, Paliouras et al. [1999] followed a dual approach by integrating stereotypes that had been acquired in a supervised learning stage and clusters acquired in an unsupervised learning stage from individual user models. As opposed to Fisher [1987], however, Paliouras et al. [1999] did not report problems stemming from the computational complexity of their clustering processes.

A computationally more manageable and currently prevalent group of algorithms for clique-based filtering follows a correlation-based neighborhood approach (see e.g. Resnick et al. [1994], Hill et al. [1995], Shardanand and Maes [1995], and Konstan et al. [1997]). Its basic idea is to first select a subset of users that are similar to the current user based on known characteristics, and to subsequently compute predictions for unknown characteristics based on a weighed aggregate of their ratings. Alternative algorithms for clique-based filtering include Bayesian networks, vector-based similarity techniques, and induction rule learning (see Breese at al. [1998] and Herlocker et al. [1999] for an overview and evaluation). Bayesian networks are reported to compute predictions faster and to have less resource requirements than correlation-based approaches. They require however a dedicated learning phase before they can be employed for clique-based filtering (i.e., they cannot learn incrementally in real time). Deployed to real-world scenarios, this implies that the "...learning phase ... can take up to several hours and results in a lag before changed behavior is reflected in recommendations" [Breese at al., 1998]. Such a system behavior, however, seems inappropriate for the usage scenario of WebGuide (see Chapter 7.1) and hardly appropriate for many other real-world user modeling scenarios as well.

We decided to employ *Spearman correlation* for determining the proximity between users in the MLC, which is based on ranks rather than values. The transformation of values into ranks uses the aforementioned classifiers in the System Model (see Figure 8-7), which replace values by class means. Although this approach requires an initial classification step for several attribute types (e.g., characters, strings), it considerably increases at the same time the number of user characteristics that can be leveraged for finding similarities (including e.g. demographic attributes like sex and age). Another advantage of Spearman correlation is that it does not rely on model assumptions (e.g., regarding the probability distribution of attribute values), as opposed to the widely used Pearson correlation [Breese at al., 1998; Herlocker et al., 1999]. Regarding coverage and accuracy[98], Spearman correlation performs similar to Pearson correlation, as long as there are a meaningful number of ranks (in general: the fewer ranks, the more information gets lost).

[98] Coverage indicates the percentage of characteristics across all users for which the system was able to produce predictions. The accuracy of predictions generated by the system can be assessed using measures e.g. from statistics (like mean absolute error and root mean square error) and decision-support (e.g., receiver operating characteristic).

The overall mentor learning process is then carried out in the following three steps [Herlocker et al., 1999]:

i. *Finding similar users*
Similarity between two users is determined by computing the (linear) Spearman correlation coefficient for the two user models (see Formula 8-6).

$$w_{a,u} = \frac{\sum_{i=1}^{m} (rank_{a,i} - \overline{rank_a})(rank_{u,i} - \overline{rank_u})}{\sqrt{\sum_{i=1}^{m} (rank_{a,i} - \overline{rank_a})^2} \sqrt{\sum_{i=1}^{m} (rank_{u,i} - \overline{rank_u})^2}}$$

Formula 8-6: Spearman correlation

$w_{a,u}$ thereby is the similarity weight for the active user *a* and a neighbor *u* based on *m* assumptions that are available for *a* and *u*. $rank_{a,i}$ represents the rank of the value of assumption *i* for user *a*. $\overline{rank_a}$ is the mean rank for all assumptions about user *a*.

In the current implementation of the MLC, the correlation coefficient is calculated for active users and all available user models. This approach seems to be appropriate for the small taxonomy of nearly 500 leaf entries and the rather small user population we anticipate for WebGuide in the near future. If the user population grows, however, such an approach can be assumed to become quickly impractical (see the results of our performance evaluation in Chapter 9.3). For further work on the MLC, we propose to restrict the search space for neighbors to a set of users. In this vein, relevant techniques that seem worthwhile to investigate include statistical sampling methods, singular value decomposition [Deerwester et al., 1990], and the employment of clusters of user profiles instead of individual users [Herlocker, 1999].

Another simplifying assumption that underlies the current implementation of the MLC is that all users and all assumptions contained in their models are treated evenly when computing the correlation coefficient. Assumptions that have been maintained by the user can e.g. be associated a higher weight than assumptions that have been acquired from usage data by the ULC. Following this, assumptions that have been predicted from stereotype models can be associated with the lowest weight. Such an approach is quite similar to the prioritization of assumptions from individual users' models over assumptions that are inherited from stereotypes in earlier user modeling work, e.g. in Kobsa and Pohl [1995]. Moreover, there are several proposals in the literature for differentiating the weighting of single assumptions. Herlocker et al. [1999] propose variance weighting, where assumptions with a high variance across the user population are emphasized and assumptions with a low variance are de-emphasized. This results in higher weights for assumptions that presumably segregate a user population more than assumptions, where users exhibit very similar characteristics. An example from the domain of WebGuide is users' interest in nightlife, which shows a much higher variance across Heidelberg tourists than users' interest in the castle of Heidelberg, which can be presumed high for all users in the population [Kaul, 1999][99]. Although quite convincing at first glance, Herlocker et al. [1999] could not report a significant effect on the accuracy of their prediction algorithm with and without variance weighting. Breese et al. [1998] propose inverse user frequency, which is motivated by the inverse document frequency measure from information retrieval [Salton and McGill, 1983]. Applied to user modeling, Breese et al. associate those assumptions that are contained in only a few user models with a higher weight than assumptions that are contained in many user models. The correlation-based approach that Breese et al. [1998] employed in their evaluation was enhanced by inverse user frequency. Moreover, they employed additional refinements like default voting and case amplification. Concluding, it seems worthwhile to further refine our approach of finding similar users, e.g. by considering the aforementioned extensions reported in the literature. In order to substantiate this, however, we highly recommend further empirical work in the Deep Map domain.

ii. *Selecting mentors*

Once the Spearman correlation coefficients have been computed for a given user, a relatively small number of most similar neighbors (the so-called *mentors*) must be selected from the set of similar users. In general, prediction accuracy increases with an increasing neighborhood. The increment in accuracy was however often found to decrease and even to turn negative at some point when more neighbors become added [Shardanand and Maes, 1995; Herlocker et al., 1999; Sarwar et al., 2000]. From a performance point of view it also seems advisable to restrict the number of potential neighbors to a reasonable number in real-world environments with millions of users.

The current version of the MLC can be configured to follow a correlation-thresholding or a best-n-neighbors approach when selecting mentors. Correlation-thresholding selects all those neighbors as mentors that have a correlation coefficient greater than a predefined threshold [Shardanand and Maes, 1995]. Depending on the situation, however, there may be too few or too many mentors remaining. The best-n-neighbors approach is to pick a fixed number of most similar neighbors as mentors. If there are only few highly correlated neighbors, this approach may select many neighbors as mentors whose correlation is low, thereby possibly reducing the prediction accuracy. And in case of many highly correlated neighbors, neighbors may be excluded that could have added to the accuracy of the prediction. The current version of the MLC for Deep Map uses the best-n-neighbors approach (with n = 20) as a default, which showed a good performance with regard to coverage and accuracy in the empirical evaluation of Herlocker et al. [1999].

[99] In other words: Heidelberg castle seems to be ‘a must’ for every visitor, as opposed to nightlife.

iii. *Computing predictions*

A variety of approaches have been discussed in the literature for computing predictions from a set of mentors. In the MLC, an assumption *i* about the active user *a* is predicted using the following deviation-from-mean approach over the selected mentors (*n* is the number of mentors, $w_{a,u}$ is the similarity weight for the active user *a* and a mentor *u*)[100].

$$p_{a,i} = \overline{rank_a} + \frac{\sum_{u=1}^{n} (rank_{u,i} - \overline{rank_u}) \cdot w_{a,u}}{\sum_{u=1}^{n} w_{a,u}}$$

Formula 8-7: Deviation-from-mean prediction

The deviation-from-mean approach takes into account that users' means may vary. Therefore, *u*'s ranks are normalized by their means, and the prediction for *a* is normalized by *a*'s means. In classical collaborative filtering scenarios with explicit ratings, this approach is motivated by the observation that an individual user's rating distributions often center around a mean, i.e. "one user may tend to rate items higher, with good items getting 5s and poor items getting 3s, while other users may give primarily 1s, 2s, and 3s. Intuitively, if a user infrequently gives ratings of 5, then that user should not receive many predictions of 5 unless they are extremely significant" [Herlocker et al., 1999]. We were not able to validate this hypothesis in data we had available from the domain of WebGuide [Kaul, 1999]. From 348 Heidelberg tourists that were asked to rate their interest in 19 topics, nearly 80% of the respondents used the whole rating scale. Nevertheless, converting mentors' interest distribution as an average from the mean into an individual user's rating distribution by adding it to the user's mean seems to be a reasonable approach anyway.

If no mentors were found for a given user, we compute the prediction as the average of the deviation-from-mean across all users, i.e.:

$$p_{a,i} = \overline{rank_a} + \frac{\sum_{u=1}^{n} (rank_{u,i} - \overline{rank_u})}{n}$$

Formula 8-8: Average deviation-from-mean prediction

[100] This approach has been taken in the original GroupLens system [Resnick et al., 1994; Konstan et al., 1997]).

This approach exhibited a good coverage and accuracy in the evaluations by Herlocker et al. [1999] and performed much better than the following simple mean across all users:

$$p_{a,i} = \frac{\sum_{u=1}^{n} rank_{u,i}}{n}$$

Formula 8-9: Average prediction

In the MLC, this simple mean is only calculated when the User Model of the current user does not contain any assumptions at all (neither ones that were explicitly provided by the user, nor ones acquired by the ULC).

In summary, the method for clique-based filtering that is employed in the MLC

i. allows for incremental learning and can thus keep up with a user's changing interests and preferences (cf. Chapter 8.4.1 for a discussion of the same quality for the ULC),

ii. represents learning results explicitly (also cf. Chapter 8.4.1 for the ULC),

iii. is able to adapt to a variety of user modeling needs (e.g., it allows designers to differentiate between mentors and stereotypes and to assign different weights, and can integrate algorithmic extensions like default voting, inverse user frequency, and case amplification [Breese et al., 1998]), and

iv. performs well in terms of predictive coverage and accuracy, responsiveness, and resource consumption.

We believe that there is further potential for synergistic effects between feature-based and collaborative filtering approaches that has not yet been fully investigated yet (for recent work on this topic, we refer to Balabanovic and Shoham [1997], Claypool et al. [1999], Condliff et al. [1999], and Mooney and Roy [1999]. Likewise the same can be anticipated for the application of clustering and partitioning algorithms to the matrix of users and content features (for recent work on this topic, we refer to Herlocker [1999] and O'Conner and Herlocker [1999]).

8.5.2 Usage Scenario

Table 8-1 shows part of the interest models of the fictitious users Joe, John, and Al. Their models already contain many assumptions about the probability of their interest in different types of buildings in Heidelberg. This information has been either explicitly provided by them or acquired by the ULC from WebGuide usage data. During design time, the user model developer added the stereotype 'Art Lover' to the set of user models available to the MLC that contains some presumably typical characteristics of art lovers. At this point it was unclear, however, whether an interest in bridges can be attributed to this user group.

Therefore, the developer decided to let the MLC predict this interest at runtime, based on the similarity of this stereotype to models of individual users. Another missing piece of information is the probability of John's interest in mansions.

Interests Users	Churches	Restaurants	Mansions	Bridges
Joe	0.80	0.30	0.90	0.70
John	0.30	0.90	***?***	0.50
Al	0.40	0.40	0.70	0.30
Art Lover	0.90	0.10	0.90	***?***

Table 8-1: Initial interest models

Before computing the Spearman correlation, the MLC has to convert the interest probabilities into ranks, thereby utilizing the classifier for the respective interests in the System Model (see Figure 8-7). Table 8-2 depicts the resulting interest models with associated means for the case in which the classifier divides the interval [0,1] into ten equal classes.

Interests Users	Churches	Restaurants	Mansions	Bridges	***Mean***
Joe	0.85	0.35	0.95	0.75	*0.72*
John	0.35	0.95	***?***	0.55	*0.61*
Al	0.45	0.45	0.75	0.35	*0.50*
Art Lover	0.95	0.15	0.95	***?***	*0.68*
Mean	*0.65*	*0.47*	*0.88*	*0.55*	

Table 8-2: Initial interest models with classified user interests

In order to find similar users, the MLC calculates the Spearman correlation coefficients for all pairs of users. The result is shown in Table 8-3.

$W_{a,u}$	Joe	John	Al	Art Lover
Joe	1.00	-0.98	0.47	0.98
John		1.00	0.18	-1.00
Al			1.00	0.50
Art Lover				1.00

Table 8-3: Spearman correlation coefficients

The next task for the MLC is to select mentors for John and Art Lover from their neighborhoods. Applying a threshold of 0.3 to the correlation coefficients, the MLC cannot find an appropriate mentor for John since Al's similarity seems too weak[101], and since Joe and Art Lover even exhibit interest probabilities that are nearly the opposite of John's. For the stereotype Art Lover, the MLC identifies Joe and Al as good mentors. John is not considered, since his correlation coefficient is below the threshold.

Finally, the prediction algorithm can be selected and predictions can be computed. Since no mentor has been found for John, the MLC calculates the prediction for his interest in mansions as a deviation-from-mean average across all users. The interest in bridges for the stereotype Art Lover can be predicted as a deviation-from-mean with Joe and Al as mentors. Table 8-4 shows the resulting complemented interest models.

Interests Users	Churches	Restaurants	Mansions	Bridges
Joe	0.80	0.30	0.90	0.70
John	0.30	0.90	***0.86***	0.50
Al	0.40	0.40	0.70	0.30
Art Lover	0.90	0.10	0.90	***0.64***

Table 8-4: Interest models including predictions

[101] The correlation coefficient lies between [–1, +1] and can be interpreted as follows: (i) a positive correlation coefficient indicates that high ranks for one user are associated with high ranks for another user (and vice versa), (ii) a negative correlation coefficient indicates that high ranks for one user are associated with low ranks for another user (and vice versa), and (iii) the closer the correlation coefficient is to 0, the weaker is the relationship.

8.5.3 Implementation

The design and implementation of the MLC follows the same principles we already discussed for the ULC (see Chapter 8.4.3). In fact, great parts of ULC's infrastructure (e.g., access to the Directory Component, internal cache management) are employed in the MLC with only minor modifications. Therefore, we focus in this sub-chapter on relevant differences between the ULC and the MLC.

In order to achieve *flexibility*, the MLC maintains a comprehensive list of parameters for controlling the learning process. Besides the more general control parameters we already presented in Chapter 8.4.3, there are several MLC-specific ones that control e.g.

- the location of relevant (sub-) models hosted by the UMS (e.g., User Models, Service Model, demographic characteristics, domain taxonomy, and classifiers within the System Model) and
- the collaborative filtering process (e.g., mentor selection strategies, various thresholds and counters).

Regarding *scalability*, we assume that the current version of the MLC linearly scales with the number of user profiles and the workload in small real-world environments (see Chapter 9.3 for related results from our experiments). We base this assumption upon our decision to not maintain the correlation matrix as a whole in main memory, but selected vectors only that represent (i) users that have been selected as mentors or (ii) users that are currently logged in to the UMS. In an additional effort, we restricted vector contents to those assumptions that are leaf entries in the domain taxonomy. The handling of the more general interests is left over as a task for the DIC, which can be assumed to require less system resources than the MLC (see also Chapter 8.6).

The current version of the MLC is searching for similar users in the whole user population. Against the enormous size of many real-world environments, this seems to be an important shortcoming (see the results of our performance experiments in Chapter 9.3). Future work may aim at overcoming this important limitation, e.g. by applying statistical sampling methods that narrow the search space to a reasonably sized sample of user profiles (see Chapter 8.5.1).

In the current version of the UMS for Deep Map, the MLC runs as a system service on Windows NT and Sun Solaris (i.e., like the ULC). The implementation language used is Java 1.2. The communication with the Scheduler is established via VisiBroker for Java and access to DS is facilitated via standard LDAP access facilities.

8.6 Domain Inferences

In this sub-chapter, we start with a brief motivation for the DIC (Domain Inferences Component), which complements the user modeling functionality provided by ULC and MLC. We present a hypothetical scenario from WebGuide that exemplifies how DIC's domain inferences are applied to user models hosted by the UMS. We close this sub-chapter with a few remarks on selected implementation details.

8.6.1 Introduction

The DIC complements the user modeling functionality provided by ULC and MLC by applying domain inferences to individual user models. User interests that were explicitly provided by users or implicitly acquired by ULC and MLC can give rise to additional assumptions about user interests that can be inferred using the hierarchical structure of interests in the domain taxonomy (see Chapter 8.2.3) as follows[102]:

- *Sidewards propagation*: if the user is interested in a minimum percentage *s* of direct sub-interests of a given interest, then the user is assumed to be also interested in the remaining sub-interests, with a probability that equals the means of the probabilities of the current sub-interests[103].
- *Upwards propagation*: if the user is interested in a minimum percentage *u* of direct sub-interests of a given interest, then the user is presumed to also hold this interest, with a probability that equals the means of the probabilities of the current sub-interests (cf. Kobsa et al. [1994]).

By using domain inferences, individual user models can become populated more quickly with additional assumptions, which allows applications like WebGuide to come up sooner with personalized information and services (e.g., personalized tour recommendations like the ones presented in Chapter 7). Personalization speed seems not only mandatory for user modeling in Deep Map, but for many other user modeling scenarios as well (e.g., in Web-based e-commerce). A second advantage of applying domain inferences to individual user models is the efficiency of this process in terms of system resources, especially when comparing resource requirements of the DIC with those of the MLC (see Chapters 8.5.3 and 9.3). Another motivation for domain inferences is related to the richness of user model contents: over time, a variety of assumptions about a user's interest are acquired by the UMS from different sources (e.g., user-supplied information, ULC, MLC including stereotypes, DIC) and put at the disposal of user-adaptive applications. This enables applications to take the origin of assumptions about users' interests for adaptation purposes into account (e.g., user-supplied information, individual probabilities of interests, normalized probabilities of interests, predicted probabilities of interests, inferred probabilities of interests). Moreover, this constitutes a valuable source of information for further refinements of user modeling functionality and for further empirical work in this area (e.g., the performance of different user modeling components in terms of accuracy and coverage).

The inference rules presented above have been adapted from those employed in the adaptive hypermedia system 'KN-AHS' [Kobsa et al., 1994]. In its user modeling component, inference rules exploit sub- and super-relationships (e.g., 'GUMS' and 'BGP-MS' are 'User Modeling Shell Systems') and 'associated-concept' relationships between fields and their respective terminology (e.g., 'BGP-MS' is associated with the concept 'Stereotype Reasoning') for inferring a user's presumable knowledge about a specific

[102] Please note that assumptions inferred by the DIC (so-called *secondary* assumptions [Kobsa et al., 2001]) are not taken into account by the ULC and MLC. The latter two focus solely on so-called *primary* assumptions for user model acquisition (i.e., assumptions that have been explicitly or implicitly provided by the user).

[103] Further (empirically driven) work on the DIC may aim at improving this rather simple algorithm, e.g., by investigating the prediction algorithms we already proposed for the MLC (e.g., deviation-from-mean, deviation-from-mean-average), including our proposals for further work on these algorithms.

concept. Relevant differences between the domain inferences implemented in the DIC and those in KN-AHS include the following:

- User modeling in KN-AHS focuses on users' terminological knowledge, whereas user modeling in Deep Map (currently) focuses (mainly) on interests and a set of demographic attributes.
- The representation sub-system of the user modeling component in KN-AHS maintains either a user's familiarity or unfamiliarity with a certain concept, whereas the Representation sub-system in the UMS for Deep Map maintains a set of probabilistic assumptions and a related classification for a user's interest.
- The domain taxonomy employed in KN-AHS employs two types of relationships, whereas the one employed in the UMS for Deep Map represents only sub-/super-relationships.
- Additional inference rules in KN-AHS take advantage of the 'associated-concept' relationships.
- Domain inferences in KN-AHS can be controlled by two percentage thresholds, whereas domain inferences in the DIC can be controlled by several configuration parameters (see Chapter 8.6.3).

8.6.2 Usage Scenario

In the following, we briefly exemplify the domain inferences carried out by the DIC using a hypothetical scenario from WebGuide. We assume that the interest part of Nathan's user model comprises assumptions about his interests in buildings and architectural styles in Heidelberg. The initial state of his profile is depicted in Figure 8-14.

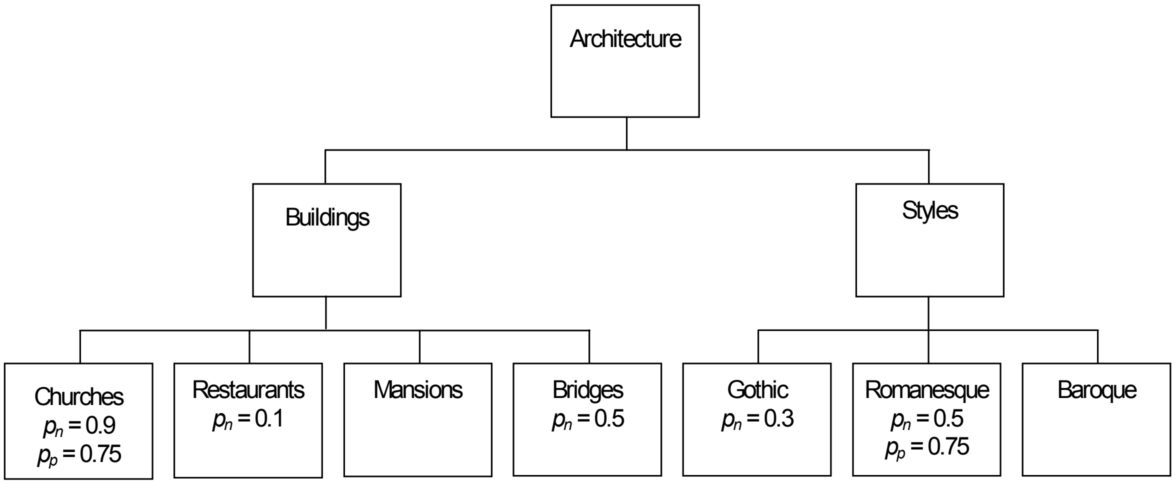


Figure 8-14: Initial state of Nathan's user model

Based on Nathan's interaction with WebGuide (he previously browsed through several documents about buildings and monuments in Heidelberg), the ULC and the MLC already acquired several normalized and predicted probabilities (abbreviated p_n and p_p) for Nathan's presumable interests. The DIC, which is subscribed to additions by the MLC (see

Chapter 8.3), now starts applying its domain inferences. This process is controlled by several configuration parameters in the Service Model, including the following ones:

- upwards propagation is enabled,
- sidewards propagation is enabled,
- a threshold of u = 60% for upwards inferences,
- a threshold of s = 75% for sidewards inferences,
- a weight of 100% is assigned to interest probabilities computed by the ULC, and
- a weight of 70% is assigned to interest probabilities predicted by the MLC.

The percentage of direct sub-concepts of Buildings in which the user is interested is 75%, which is equal to s and greater than u. Since both upwards and sidewards inferences are enabled, the DIC calculates the probability of Nathan's interest (abbreviated p_i for inferred probability) in Buildings and Mansions as follows:

$$p_i = \frac{\frac{0.9+(0.75\cdot 0.7)}{1+0.7}+0.1+0.5}{3} = 0.47$$

Formula 8-10: Inferring Nathan's interest in buildings and mansions

The percentage of direct sub-concepts of Styles in which the user is interested is approximately 66%, which is smaller than s but greater than u. The DIC therefore does not compute a probability for Baroque style. Nathan's presumable interest in Styles, however, is calculated as follows:

$$p_i = \frac{0.3+\frac{0.5+(0.75\cdot 0.7)}{1+0.7}}{2} = 0.45$$

Formula 8-11: Inferring Nathan's interest in styles

Finally, the percentage of direct sub-concepts of Architecture in which the user is interested is 100%. The DIC calculates the presumable interest in Architecture as a simple means of the inferred interests in Buildings and Styles and updates the user model accordingly. Figure 8-15 depicts the final state of Nathan's user model.

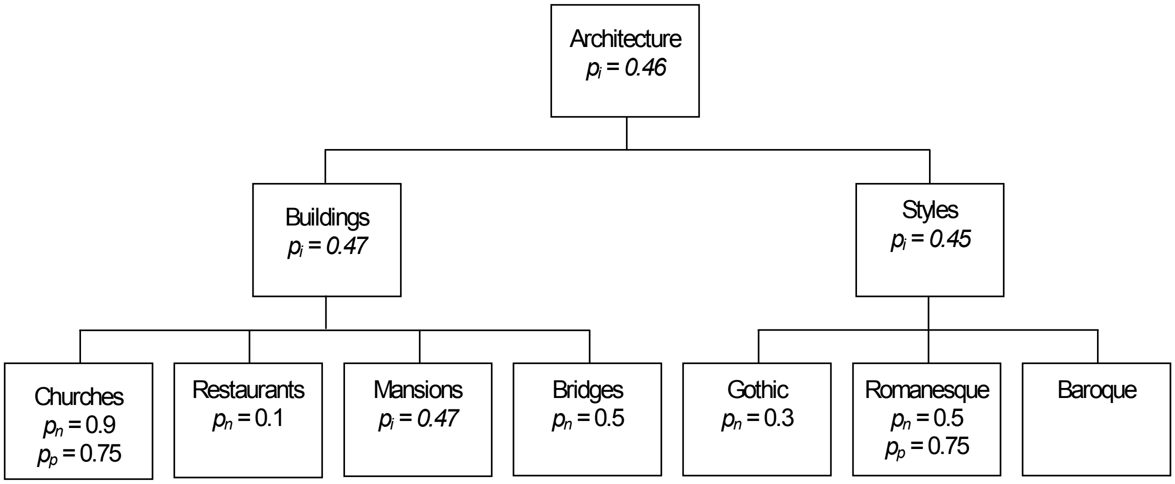


Figure 8-15: Final state of Nathan's user model

8.6.3 Implementation

The design and implementation of the DIC follows the same principles we already discussed for the ULC and the MLC. In the remainder of this sub-chapter, we focus on a few relevant differences between the DIC and the other two learning components.

In order to provide *flexibility*, the DIC can be controlled via the more general parameters we already presented for the ULC in Chapter 8.4.3 and via several DIC-specific ones, which control e.g.

- the location of relevant (sub-) models hosted by the UMS (e.g., User Model, Service Model, domain taxonomy within the System Model) and the
- process of drawing domain inferences (e.g., propagation direction, propagation thresholds, interest weights).

Regarding *scalability*, we assume that the current version of the DIC linearly scales with the number of user profiles and the workload in small and medium-sized real-world environments. For related feedback from our performance experiments, we refer to Chapter 9.3. Depending on the system configuration, however, performance problems may occur in case of the propagation speed in (sparsely populated) user models being too high (e.g., if the thresholds for upwards and sidewards propagation are too low). Such problems should not persist, however, since once a user model has been populated, the DIC henceforth updates the user model from its internal cache only if an inferred probability has significantly changed. The resource requirements of the DIC can be assumed moderate.

The implementation environment for DIC, ULC, and MLC is the same (i.e., Java 1.2., communication with the Scheduler via VisiBroker for Java, and access to DS via standard LDAP access facilities).

IV
Evaluation and Discussion

In the following chapter, we present selected results of the performance evaluation that we conducted. The main aim of these experiments was (i) to obtain empirical evidence regarding the runtime behavior of our server in small and medium-sized application environments, and (ii) to point out potential strengths as well as limitations of its design and implementation. In the subsequent chapter, we revisit our server and user modeling requirements again and discuss our server against this background. In the final chapter of this thesis work, we summarize our main findings and briefly present some lessons learned from deploying our server to real-world environments. Regarding scalability, a large-scale test with several million user profiles and a workload that is representative of major Web sites confirms that the user modeling performance of our server will not impose a significant overhead for a personalized Web site.

9 User Modeling Server: Experiments

The main aim of our experiments is to experimentally test the runtime behavior of our user modeling server under realistic workload conditions to ascertain its satisfactory performance in the target application environments. Based on empirical Web usage research, we develop a model of real-world workload that allows us to simulate workload conditions that closely resemble the target application environments. More specifically, we simulate users' interaction with a hypothetical personalized Web site to test the performance of our user modeling server under different workload conditions. The content of each Web page is characterized by 1-3 terms taken from the Deep Map Taxonomy. Web page requests by a user lead to add and query operations in her user profile on the user modeling server: the terms of the requested Web page are processed and added to her interest model, and the user's interests in terms of the domain taxonomy are queried to personalize the requested Web page. As a shortcut though, we omit the Web server and application server in our simulation and represent Web pages by their characteristic terms only.

This approach is in sharp contrast to the one taken in the few existing performance studies of user modeling servers (e.g., VanderMeer et al. [2000], Datta et al. [2001]) and of directory servers (e.g., Keung and Abbott [1998], Wang et al. [2000]), who all employed synthetic workload models that are not based on empirical findings about Web usage behavior.

In the first part of this chapter, we briefly describe our model of real-world workload. In the second part, we describe the test bed that we developed, and thereafter the simulated real-world Web site and the measures we recorded during our tests. After that, we describe our hardware and software configuration and outline our testing procedure. In the third part of this chapter, we discuss selected results and their implications on performance and scalability on a global server level (i.e., black box perspective) and on the level of single User Modeling Components (i.e., white box perspective). Based on our findings and the testing approach chosen, we point out potential strengths as well as weaknesses in the current design and implementation of our UMS.

9.1 Model of Real-World Workload

A model of real-world workload for user modeling servers fundamentally hinges on empirical findings about Web usage behavior. In the literature on Web traffic characterization and prediction, however, such findings are rather scarce. Most of the work reported in the literature has been based on traffic data from relatively small departmental servers, typically at universities (e.g., Almeida et al. [1996], Arlitt and Williamson [1996], Zukerman et al. [1999]). These data however may not be very representative for the typical online behavior of Web users, mainly for a combination of two reasons:

i. The bulk of Web traffic today is served by large commercial Web sites like MSNBC [2001], CNN [2001], and Yahoo! [2001]. Computer Scope [1999], e.g., found that 35% of users' surfing time is spent on merely 50 (commercial) sites.
ii. The user audience at these more typical sites is quite different in terms of size and online behavior from the one visiting the small departmental Web servers that have been analyzed in previous work (cf. Padmanabhan and Qiu [2000]).

Another drawback of many characterizations and predictions of Web traffic is the fact that they are based on proxy logs[104] (e.g., Duska et al. [1997], Gribble and Brewer [1997], Wolman et al. [1999a], Wolman et al. [1999b]) or Web server logs (e.g., Almeida et al. [1996], Padmanabhan and Qiu [2000]). The information contained in these logs seems especially useful for evaluating and improving caching and prefetching strategies. For analyzing usage behavior, however, both types of logs are of rather limited value since they do not reflect all communication that takes place between browsers and Web servers. For instance, clients may connect to Web servers via several proxies, and numerous caches may affect the amount of traffic between browsers and Web servers.

In order to avoid these limitations, we adopted findings from recent empirical research on usage behavior on the Internet. We expect that these findings constitute a more promising basis for our model of real-world workload than those mentioned before, since they provide an authentic view of users' online behavior, as opposed to the keyhole perspective of previous proxy- and server-based studies.

Rozanski et al. [2000] recently conducted a comprehensive analysis of click-stream data collected by 'Nielsen//NetRatings'[105]. The data was collected at the *client side* from a panel of 2,466 Internet users between July and December 2000. In a first step, Rozanski et al. identified 186,797 user sessions. They defined a session as the total time from when a user signs on to the Internet to when she signs off, or to the point when her activity ceases for more than an hour. All Web page requests induced by a user during that time became part of a user session. In a subsequent step, Rozanski et al. tested a variety of session characteristics (e.g., session length, number of Web sites visited) with regard to their

[104] Proxies aim at improving the communication between a set of clients and Web servers, mainly by caching and/or prefetching frequently requested Web contents.

[105] Nielsen//NetRatings is an audience measurement service provided by Nielsen Media [2000] and NetRatings [2000]. Their U.S. media research panel is one of the largest in the world and comprises 62,000 users at home and 8,000 users at work.

suitability for clustering user sessions. The most differentiating session characteristics they found were the following:

- *Session length* is defined as the length of a user session on the Internet.
- *Time per page* denotes the time between consecutive Web page requests.
- *Category concentration* is defined as the percentage of time a user stays at Web sites of the same category (e.g., news, sports, entertainment, real estate).
- *Site familiarity* is determined by the percentage of time a user stays at familiar sites, i.e. sites she had previously visited four or more times.

Based on these session characteristics, Rozanski et al. carried out a cluster analysis and identified the following patterns of Web usage:

- *Quickie* sessions are short (1 minute) visits to 1 or 2 familiar sites, to extract specific bits of information (e.g., stock quotes, sports results). Users visit 2.2 pages per site on average, and spend about 15 seconds on a page.
- *Just the Facts* sessions aim at finding and evaluating specific pieces of information at related sites (e.g., compare product offers). Sessions last 9 minutes on average. Users visit 10.5 sites and 1.7 pages per site, with about 30 seconds per page.
- *Single Mission* sessions focus on gathering specific information or completing concrete tasks (e.g., finding the Web site of a scientific conference and registering for it). They visit 2 Web sites on average, which belong to the same category (e.g., search engines or portals). Users quite carefully read the content of (frequently unfamiliar) Web pages in approximately 90 seconds. The average session length is 10 minutes, and 3.3 pages per site are being visited.
- *Do It Again* sessions are strongly focused on sites with which the user is very familiar (e.g., online banks, chat rooms). Users stay approximately 2 minutes at each Web page. The average session length is 14 minutes, with 2.1 visited sites and 3.3 page requests per site.
- *Loitering* sessions visit familiar 'sticky' sites, such as news, gaming, telecommunications, and entertainment. Sessions last 33 minutes, with 8.5 sites and 1.9 pages per site being visited (2 minutes per page on average).
- *Information Please* sessions gather broad information from a range of often unfamiliar Web sites belonging to several categories (e.g., they collect facts about a specific car model, find a dealer, negotiate a trade-in, and arrange a loan). Users visit 19.7 Web sites and 1.9 pages per site. The average session length is 37 minutes, and pages are viewed for 1 minute on average.
- *Surfing* sessions appear random, with users visiting nearly 45 sites in 70 minutes on average (about 1 minute per page and 1.6 pages per site).

Over time, users can engage in several, if not all, session types depending on how different their task contexts are. Rozanski et al. found, e.g., that two-third engaged in five or more session types and 44 percent in all seven session types. Table 9-1 summarizes the aforesaid by listing the defining and complementary characteristics of all session types. Their relative frequencies within the 186,797 distinct user sessions are depicted in Figure 9-1.

	Defining characteristics				Complementary characteristics		
Variables **Types**	**Session length**	**Time per page**	**Category concen-tration**	**Site famil-iarity**	**Number of sites**	**Pages per site**	**Time per site**
Quickies	1 min	15 sec	90%	90%	1.8	2.2	0.6 min
Just the Facts	9 min	30 sec	47%	88%	10.5	1.7	0.9 min
Single Mission	10 min	90 sec	85%	11%	2.0	3.3	4.9 min
Do It Again	14 min	120 sec	87%	95%	2.1	3.3	6.7 min
Loitering	33 min	120 sec	66%	90%	8.5	1.9	3.9 min
Information Please	37 min	60 sec	41%	14%	19.7	1.9	1.9 min
Surfing	70 min	60 sec	26%	85%	44.6	1.6	1.6 min

Table 9-1: Internet session types (based on Rozanski et al. [2000])

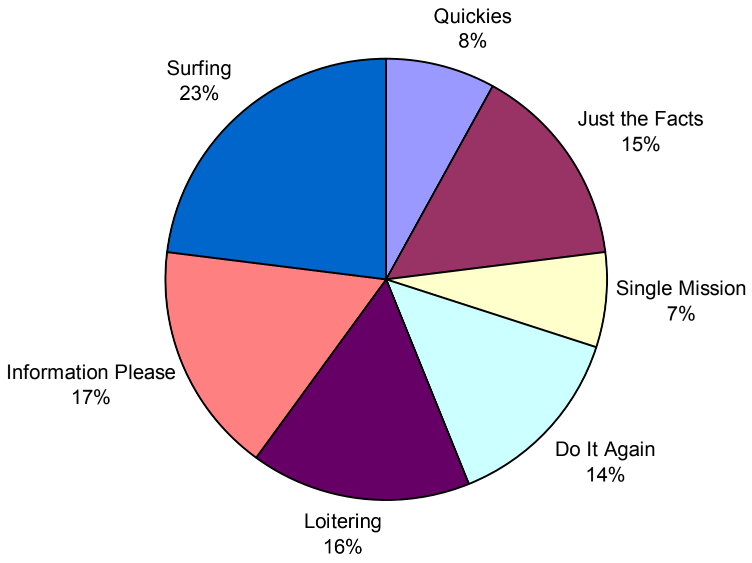


Figure 9-1: Frequency of Internet session types (based on Rozanski et al. [2000])

In order to further refine our model of real-world workload, we assume Zipf-like distributions of the frequencies in which

- terms from the Deep Map Taxonomy (see Chapter 8.2.3) become characteristic terms for Web pages,
- users engage in new sessions with our hypothetical Web application, and
- Web pages are requested by users (i.e. 'page popularity').

Our first assumption is based on the fact that term frequency distributions in documents tend to follow Zipf's law [Zipf, 1949], which expresses the fact that in general few words occur extremely often (e.g., 'the', 'and'), many words quite often (e.g., 'client', 'server'), and most words hardly ever appear (e.g., 'Zipf', 'User Modeling Server'). The second assumption is an estimate based on several studies regarding the frequency and duration of people's Internet usage (e.g., Patrick and Black [1996]). Our last assumption is based on the observation that Web page popularity tends to follow a Zipf-like distribution $1/i^{\alpha}$, where i is the popularity rank of the Web page and α an adjustment for the server environment and the domain [Nielsen, 1997; Breslau et al., 1999; Allaire and Bestavros, 2000; Padmanabhan and Qiu, 2000]. Regarding an appropriate α, we follow Padmanabhan and Qiu [2000] who analyzed the MSNBC news site and recommend an α between 1.4 and 1.6. We therefore opted for α=1.5 and use this value for all three distributions.

We assume further that our UMS has to process the following user modeling operations for personalizing a requested Web page[106]:

- *Three LDAP search operations* with Zipf-distributed terms from the Deep Map Taxonomy, namely for personalizing the page header (e.g., user-tailored banner advertisements), the navigation section (e.g., personalized links), and the content part (e.g., personalized news and product recommendations). We assume that one search is an exact one (i.e., it contains fully specified terms from the Deep Map taxonomy in the LDAP search filter) and two are substring searches (i.e., they contain randomly right-truncated terms from the Deep Map Taxonomy plus a wildcard character in the LDAP search filter). All search operations take place over the cn attribute. Examples are `(cn=Umweltbelastung)` and `(cn=Umwelt*)`, respectively.
- *One LDAP add operation* for communicating a user's hyperlink selection as an interest event to the UMS. Each event characterizes the hypermedia document selected by a random number of up to three Zipf-distributed terms from the Deep Map Taxonomy. For instance, the keywords `Umweltbelastung`, `Fauna`, and `Flora` describe a user's request of a document about the environmental impacts of tourism (see also Chapter 8.2.2).

In the following sub-chapter, we present the test bed that we developed for simulating our real-world workload and describe the testing procedure we followed in our experiments.

[106] In contrast, many personalized Web sites do not provide personalized information and services on all Web pages, which lowers the load of the UMS.

9.2 Test Bed

9.2.1 Overview

Figure 9-2 depicts our test bed. On the right side, we see the UMS for Deep Map. Below its Directory Component, several models are shown that constitute the representational basis of the UMS, namely the *User Model*, *Usage Model*, *System Model*, and *Service Model* (see Chapter 8.2). We retained the latter three models from the Deep Map project without modification (i.e., including the taxonomy described in Chapter 8.2.3), but varied the size of the User Model in our experiments, since it can be considered as an important server workload factor. We further discuss this in Chapter 9.2.2.

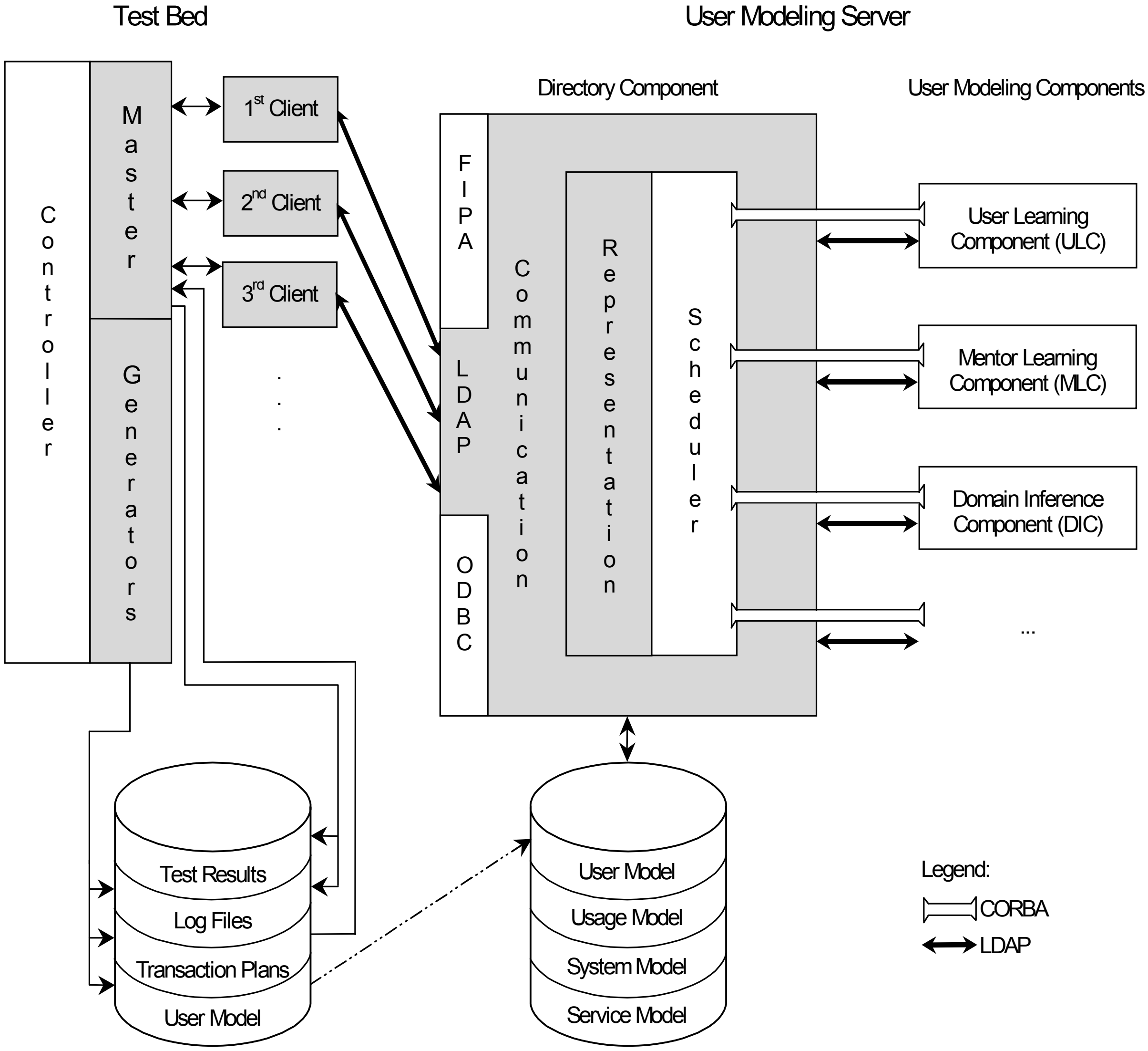


Figure 9-2: Overview User Modeling Server test bed

On the left side of Figure 9-2, we see the components that constitute our Test Bed, namely the *Controller*, *Generators*, *Master*, and *Clients*. In the following, we briefly describe the main tasks of each component. In order to facilitate orientation, we list components' names as run-in headings in italics.

Controller. Its main tasks are

i. creating the different experimental workload conditions (by, e.g., generating and initializing the required number of simultaneously operating Clients, and the number of user profiles hosted by the UMS),
ii. executing test cases within given constraints (e.g., test runtime and ratio of different types of LDAP operations), and
iii. collecting and recording client-side measures (e.g., mean response times for LDAP add operations, average number of entries affected by LDAP search operations).

In order to achieve this, the Controller employs all of the following components.

Generators create

i. User Model contents (i.e., a preset number of user profiles whose object type is `person`, `organizationalPerson`, or `inetOrgPerson`),
ii. Transaction Plans, which specify the mix of LDAP operations to be sent to the UMS, and
iii. Log Files, which contain various information about the generation processes.

Attributes in the demographic part of the generated user profiles are initialized with values that are randomly selected from lists of permissible attribute values (e.g., from a list of valid surnames or a list of U.S. ZIP codes). The interests part of the generated user profiles is initially empty. The generation of Transaction Plans can be controlled by a variety of parameters (such as the ratio of exact vs. substring LDAP searches, or the number of LDAP operations that are being submitted to the UMS during a session). The selection of (i) users from the set of generated user profiles and (ii) interests from the Deep Map Taxonomy is controlled by our Zipf distribution.

For an example of the operation of generators, we start with an excerpt of the user model `Peter Smith` (in LDIF format):

```
dn: cn=Peter Smith, cn=User Model, ou=UMS, o=gmd.de
uid: Peter Smith
userpassword:: e1NIQX1XMmxYVDErTUp3VHhpdCtDdSt2b09xS3BPU0E9
objectclass: top
objectclass: person
sn: Smith
cn: Peter Smith
```

A transaction plan for a single session of `Peter Smith` that is generated within the constraints specified by the Controller could then look as follows (in Directory Mark format, see Mindcraft [2001]):

```
bind_as
cn=Peter Smith,cn=User Model,ou=UMS,o=gmd.de
htimSm
ldap_search
cn=Peter Smith,cn=User Model,ou=UMS,o=gmd.de
LDAP_SCOPE_SUBTREE
(cn=Umwelt*)
# Rest of search operations removed
ldap_add
dn:’cn=20010123_150841_0,cn=DMI Events,cn=Usage Model,ou=UMS,o=gmd.de’
objectclass: ’top’ ’dmi event’
cn: ’20010123_150841_0’
userid: ’Peter Smith’
keywords: ’Umweltbelastung’
# Rest of search and add operations removed
ldap_unbind
```

This transaction plan specifies that a Client should log in to the UMS as `Peter Smith`, using the LDAP bind operation. The password for authentication is provided in the third line. After that, a recursive search with the filter template `(cn=Umwelt*)` is to be submitted that starts from the base entry of `Peter Smith`. The remaining search operations are omitted for reasons of brevity. The next part of this transaction plan specifies the submission of an interest event for `Umweltbelastung` to the usage model of Peter Smith. After some more searches and additions, a final unbind operation terminates the session.

Master. Its main tasks are

i. starting and initializing a preset number of Clients, each with a dedicated transaction plan,

ii. managing Clients at the time of testing, and

iii. compiling Clients’ individual performance measures into a single uniform report.

Clients execute their transaction plan (thereby submitting and monitoring LDAP requests), and report their performance measures back to the Master.

For implementing Generators, Master, and Clients, we took advantage of 'Directory Mark', a benchmark suite for LDAP servers from Mindcraft [2001][107]. Directory Mark simulates clients that simultaneously access an LDAP server and reports a variety of performance indicators, including latency and throughput. Latency refers to the time the server needs to complete LDAP operations of a specific type (e.g., mean response time for search operations). Throughput is the number of LDAP operations of a specific type that the UMS completes within a period of time (e.g., average number of add operations per second). All performance indicators reported by Directory Mark are measured from a client's point of view. Therefore, they do not only indicate the performance of the UMS, but also the performance of the network and, to a limited degree, the performance of the client computer. Integrating Directory Mark into our Test Bed was fairly easy, due to the compliance of our user modeling server with established LDAP standards. Only a few modifications had to be applied to Directory Mark, which were mainly motivated by (i) our user modeling extensions to standard LDAP object types (e.g., regarding interests and preferences) and (ii) the necessity for submitting interface events to the UMS. These modifications were realized by a wrapper around Directory Mark. This allowed us to inject event submissions with randomly generated numbers of Zipf-distributed terms from the Deep Map Taxonomy into the Transaction Plans generated by Directory Mark. These plans can then be executed by standard Directory Mark Masters and Clients.

9.2.2 Workload Simulation

So far, we described our model of real-world workload, a hypothetical Web usage scenario, and the means our Test Bed offers for simulating it (e.g., Transaction Plans). Our first experiment was a two-factor design with the following parameters:

- *number of user profiles*: 100, 500, 2,500, and 12,500;
- *number of Web page requests* per second: 0.5, 1, 2, and 4.

The first parameter refers to the number of user profiles hosted by our UMS. The overall user population of a Web site can be assumed to be considerably larger in size, since in general only a certain percentage of visitors opt for personalization[108]. The second parameter determines the number of page requests to our hypothetical Web application and, by derivation from the workload assumptions introduced in Chapter 9.1, also the number, type, and complexity of LDAP operations to be processed by the UMS. In the remainder of this dissertation, we use the term 'small and medium-sized workload' as referring to a workload that is not higher than four Web page requests per second on average[109].

[107] Directory Mark seems to be the only benchmark suite that is currently available for LDAP servers.

[108] A large German news portal estimates that merely 5% of their user population registers and actively benefits from personalized services [personal communication]. Applying this rate to our maximum number of 12,500 user profiles leads to an overall user population of 250,000. Other personalization rates for active users reported in the literature include 25% in an early version of AltaVista [Schrock, 1999] and 64% for Excite [2002]. Applying these rates to our maximum number of 12,500 user profiles leads to a total user population of about 50,000 and 19,500, respectively.

[109] Based on the Web traffic statistics of IVW [2001] for November, one can estimate that three of four German Web sites receive not more than four page requests per second on average.

The four values for each of our two parameters lead to 16 different test cases. For each value combination, we generated a test plan containing N user profiles (i.e., 100, 500, 2,500, and 12,500). To avoid starting a test run with all user profiles being empty, we introduced a warm-up phase during which the profiles became initially populated. In order to obtain (nearly) the same filling degree in all test cases, we increased the warm-up time proportional to the number of user profiles (i.e., 10 minutes for 100 user profiles, 50 minutes for 500 user profiles, etc.). The simulated Clients that are associated with these profiles are partitioned into seven classes. Each class *i* represents one of the aforementioned session types and comprises c_i Clients. Every Client exhibits the Web page request behavior that is characteristic of her class. The c_i Clients of a particular class *i* simulate a total workload of w_i page requests per second. The total workload W of all C Clients corresponds to the preset frequency of page requests (i.e., 0.5, 1, 2, or 4 per second), the ratio w_i/W approximates the observed type frequency of each class[110].

Table 9-2 shows the test plan for the workload of W=2 page requests per second. In the left part, we see the seven session types from Table 9-1, along with their relative frequencies from Figure 9-1 and the viewing times per page from Table 9-1. Quickies, for example, represent 8 percent of all sessions and spend just 15 seconds for skimming each requested Web page. The right part of Table 9-2 shows the numbers c_i of Clients that are needed to generate target workloads of w_i page requests per second. Regarding Quickies, e.g., we employed c_i=2 Clients with a total workload of w_i≈0.13 page requests per second in our Test Bed. For simulating all seven session types with a total workload of W=2 Web pages per second, we employed seven differently configured test environments with a total number of C=146 Clients.

At runtime, each Quickie client consecutively simulates a session as follows (see also the transaction plan we presented earlier for `Peter Smith`):

- log in (i.e., LDAP bind) to the UMS[111],
- simulate a Web page request (i.e., submit three LDAP search operations and one LDAP add operation as described earlier),
- wait for 15 seconds,
- simulate another Web page request,
- wait again for 15 seconds, and finally
- log off from the UMS[111].

[110] Since we do not have an empirical distribution of the session types from Table 9-1 for the level of single Web sites or types of Web sites at our disposal, we adopt for our hypothetical Web site the distribution of session types from the panel level without any further modification (e.g., normalizing the distribution according to the different numbers of requested Web pages). This selection is also motivated by empirical evidence regarding the frequency of Quickies and Just the Facts sessions we obtained from deployments of our server to several German news sites (see e.g. Fink et al. [2002]). With regard to the resulting workload, this means that our server has to provide its user modeling services also for Quickies and Just the Facts sessions (which can be regarded rather demanding, due to their short viewing times and session lengths). As opposed to that, the relative percentage of these session types can be regarded lower for many personalized Web sites, which in turn lowers the load of the UMS.

[111] We assume that many Quickie applications (e.g., retrieval of stock quotes or sports scores) will acquire and authenticate users' credentials (at least semi-) automatically (e.g., via cookies, smartcards, and/or passwords) and automatically handle user logins/logoffs by forwarding users' credentials to the user modeling server.

	Session type characteristics		Test bed settings	
Variables Types	**Relative frequency**	**Time per page**	**Number of Clients***	**Page requests per second***
Quickies	8%	15 sec	2	0.13
Just the Facts	15%	30 sec	9	0.3
Single Mission	7%	90 sec	13	0.14
Do It Again	14%	120 sec	34	0.28
Loitering	16%	120 sec	39	0.33
Information Please	17%	60 sec	21	0.35
Surfing	23%	60 sec	28	0.47
	100%		***146***	***2***

Table 9-2: Test composition for 2 Web page requests per second (*=figures rounded)

In all our tests, Clients send their LDAP operations synchronously to the UMS. This means that after invoking an LDAP operation, they wait until this operation has been completed and the reply has been received from the server. In real-world applications, waiting is often inadequate though, e.g., when a client has to respond to user interface events. Using asynchronous LDAP operations may reduce waiting and improve the overall performance of a client, especially when several threads of operations can be concurrently managed.

9.2.3 Measures

During our tests, we recorded and collected 269 measures for Clients and components of our UMS. In the following, we provide a brief overview of the most important ones:

- Measures related to test cases as a whole include:
  - *Parameter values* (i.e., number of user profiles and number of Web page requests per second).
  - *Topography* of the Test Bed and the *UMS* (i.e., single platform, multi-platform).
  - *Hardware* used for clients and server(s).

- Measures related to Clients include:
  - *Search performance*: number of search operations, minimum and maximum search time, mean time, standard deviation, number of entries affected by searches.
  - *Add performance*: number of add operations, minimum and maximum response time, mean time, standard deviation, number of entries added to user profiles.
- Measures for every UMS component (i.e., DS, ULC, MLC, DIC) include:
  - *Configuration*: process priority[112], type of events to be processed, maximum loop delay[113], minimum event counter[114], LDAP client settings (e.g., protocol version, size and time limits).
  - *Performance*: overall test duration, queue statistics (e.g., number of events processed; mean, minimum, and maximum processing time, and its standard deviation), LDAP statistics (e.g., number of search operations; mean, minimum, and maximum search time, and its standard deviation; number of user model entries affected; number of failed operations).
  - *Resources*: amount of CPU time and main memory used for the startup phase and for the whole test run.

Most of these measures were automatically recorded by our Test Bed in several Log Files.

9.2.4 Hardware and Software Configuration

We tested our UMS in a single platform and a multi-platform deployment scenario. In the single platform scenario, the whole UMS (i.e., Directory Component, ULC, MLC, and DIC) was running on a single computer. The server hardware we used can be regarded typical for small and medium-sized application environments: a PC with two 800 MHz processors, 1 GB RAM, a RAID controller with two 18.3 GB UW-SCSI hard disks, and a network card running at 100 MB per second. The software used was Windows NT 4.0, iPlanet Directory Server 4.13, and VisiBroker 3.4. The learning components of our UMS were compiled with Java 1.2.2 and took advantage of the Java Hot Spot Server Virtual Machine 2.0. In the multi-platform scenario, we installed only the Directory Component on the aforementioned dual processor computer. The User Modeling Components were deployed to additional PCs as follows:

- ULC was installed on a PC with an 800 MHz processor, 256 MB RAM, and a network card running at 100 MB per second. The operating system was Windows NT 4.0.
- MLC was running on a PC with a 600 MHz processor, 384 MB RAM, and a network card running at 100 MB per second. The operating system was Windows NT 4.0.
- DIC was deployed to a PC with a 600 MHz processor, 256 MB RAM, and a network card running at 100 MB per second. The operating system was Windows NT 4.0.

[112] This parameter refers to the process priority on the operating system level.

[113] This parameter defines the maximum amount of time a component should wait for incoming events before starting a new processing loop.

[114] This parameter specifies the minimum amount of events necessary for a component to start a new processing loop.

It is important to note that these low-end PCs were not homogeneously equipped. Future testing rounds may aim at overcoming this anomaly.

Regarding software configuration, we changed a few standard parameters of DS as follows:

- *Threadnumber* was changed to 60. This setting instructed DS to create 60 threads during startup for simultaneously processing incoming requests.
- *Maxthreadsperconn* was set to 60. This value allowed each LDAP connection to use up to 60 threads at runtime, if necessary.
- *Db_durable_transactions* was disabled. This setting disabled logging of recovery information.
- *Cachesize* was set to approximately 250,000 for the main database and to 2,000 for the event database. These values instructed DS to maintain the aforementioned numbers of LDAP entries in cache memory.
- *Dbcachesize* was set to approximately 200 MB for the main database and to 1.8 MB for the event database.

Finally, we present the most important configuration settings of our User Modeling Components:

- *Deleteeventsafterprocessing* was set to true. This setting enabled the deletion of processed interest events in the Usage Model.
- *Lazyupdate* was set to 0.1. This setting instructed our learning components to persist a newly calculated interest probability only when it differs more than 10% from the value stored in the user profile.
- *Maxinterests* and *maxpredictions* was set to 6. This value instructed the ULC and the MLC to maintain in every user profile the 6 most evident interests and preferences only.

Furthermore, we restricted the amount of memory that is available for the DIC and the ULC to 100 MB, and the memory for the MLC to 150 MB. All components were running with the same priority on the operating system level.

Our Test Bed was installed on a single PC with an 800 MHz processor, 512 MB RAM, and a network card running at 100 MB per second. The operating system was Windows NT 4.0, and our Test Bed employed DirectoryMark 1.2.

9.2.5 Testing Procedure

In all experiments, we adhered to the same sequence of procedures:

1. Generate a User Model (which comprises a predefined number of user profiles) and Transaction Plans for every Client group (each group exhibits the page request behavior of one session type).
2. Populate all user profiles in an initial warm-up phase (the runtimes for each test case ranged from 10 minutes for 100 user profiles to 1,250 minutes for 12,500 user profiles).
3. Reboot the servers.
4. Run each test case for 300 minutes.

Finally, we collected the aforementioned measures from the Log Files generated by our Test Bed and compiled them into a single file.

9.3 Evaluation Results

In the following, we present selected results from the evaluation of our user modeling server. To guide our presentation and discussion, we start with a black box perspective that represents a client's point of view, and then switch to a white box-perspective where we analyze the behavior of the User Modeling Components in more detail. In order to facilitate orientation, we note the different testing scopes (i.e., Web application, ULC, MLC, DIC) as run-in headings in italics.

9.3.1 Black Box Perspective

9.3.1.1 Performance and Scalability

Web application. Figure 9-3 depicts the mean response times of our UMS for processing the four user model operations that personalize a Web page from the viewpoint of our hypothetical application (namely three LDAP search and one add operations, see Chapter 9.1). Mean times are shown for all 16 test cases.

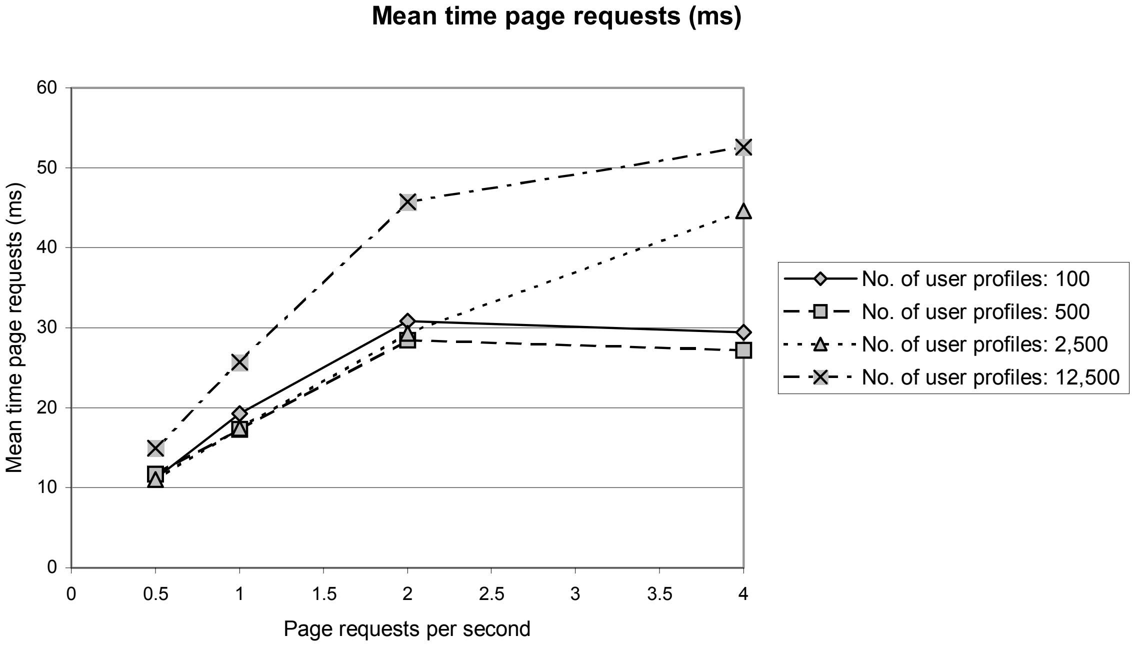


Figure 9-3: Mean time User Modeling Server page requests

In general, mean times increase only degressively with the number of page requests and user profiles. In two cases (namely for 100 and 500 user profiles), the mean times for 4 page requests per second are even slightly lower than those for 2 page requests per second. This advantageous response time behavior is mainly due to database caching in the Directory Component (see also Figure 9-4 and the related discussion). The more user model operations are sent to the server for a given number of user profiles, the faster this cache gets filled with entries of different popularity (see our Zipf distribution in Chapter 9.1) and the more operations can therefore be directly served from cache memory, thereby avoiding rather costly accesses to secondary storage. Another effect that is related to database caching can be observed for a given number of page requests and varying numbers of user profiles: the mean times for 100, 500, and 2,500 user profiles are quite similar (except for 2,500 users and 4 page requests per second), but the mean times for 12,500 users are notably higher. We assume that this behavior results mainly from a higher hit rate (i.e., a higher probability that a specific piece of information is contained in cache memory) in test cases with smaller numbers of user profiles. High numbers of user profiles lead to more frequent accesses to secondary storage, with negative effects on the performance of the server.

In a small to medium-sized application scenario, the overall performance and scalability of our UMS seems as highly satisfactory. The mean response times for processing four user model operations and returning the results to up to 288 Clients in parallel (in the case of 4 page requests per second) is always smaller than 53 milliseconds. From a usability point of view, such a low user modeling overhead should enable a personalized Web-based application to provide its services within the desirable response time limit of 1 second and, in any case, below the mandatory limit of 10 seconds [Nielsen, 1993][115]. The moderate surge of the mean response time when the number of clients and user profiles increases does not suggest looming performance and scalability limits.

The above figures reflected the performance of our server for a mix of three LDAP search and one add operations. In the following, we look at the results for each type of operation separately.

Searches. Figure 9-4 depicts the mean response times of our server when processing LDAP search operations from the viewpoint of our hypothetical Web application. The search performance and scalability resembles very much the one in Figure 9-3. As noted earlier, this response time behavior is mainly due to the generously sized database cache (see Chapter 9.2.4). The Directory Component was seemingly able to serve many search requests directly from cache memory, especially when many Clients access our server simultaneously. With all mean response times being smaller than 15 milliseconds (even in the case of 288 simultaneous Clients), the search performance of our server seems excellent.

Additions. Figure 9-5 depicts the mean response times of our server for processing LDAP add operations from the viewpoint of our hypothetical Web application. They increase only degressively with the number of page requests, and even decrease for 100 user profiles and 4 page requests per second. The mean response times for 100, 500, and 2,500 user profiles appear quite similar.

[115] A response time below one second allows most users to retain their flow of thought. Ten seconds is about the limit for most users keeping their attention focused on the dialogue with an application system (cf. Nielsen [1993]).

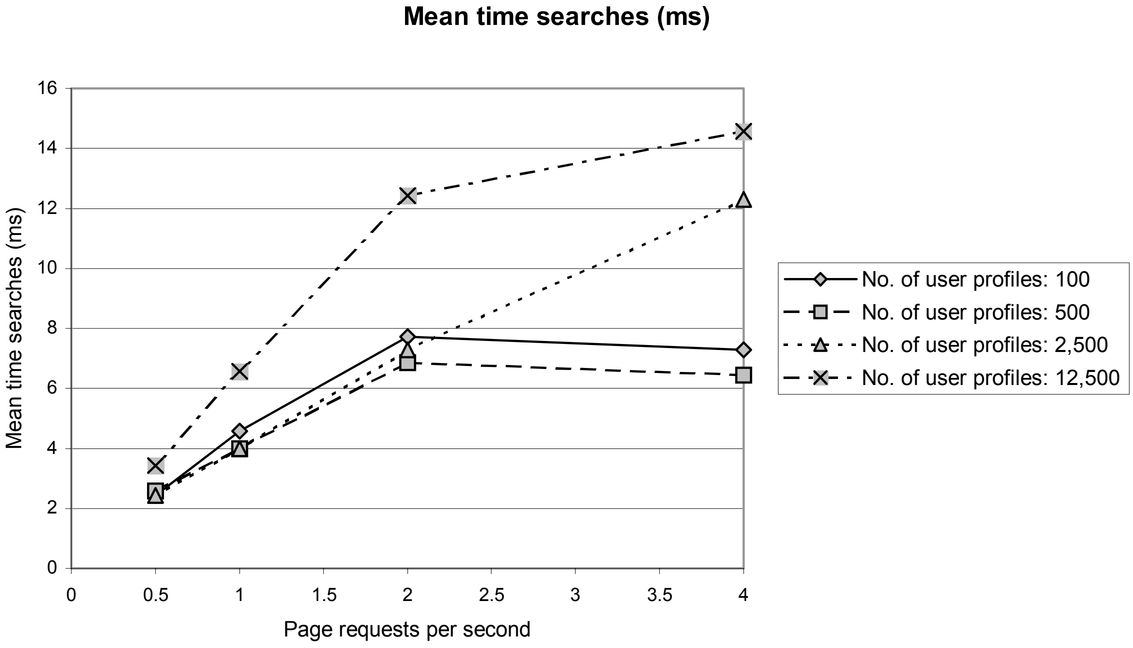


Figure 9-4: Mean time User Modeling Server search operations

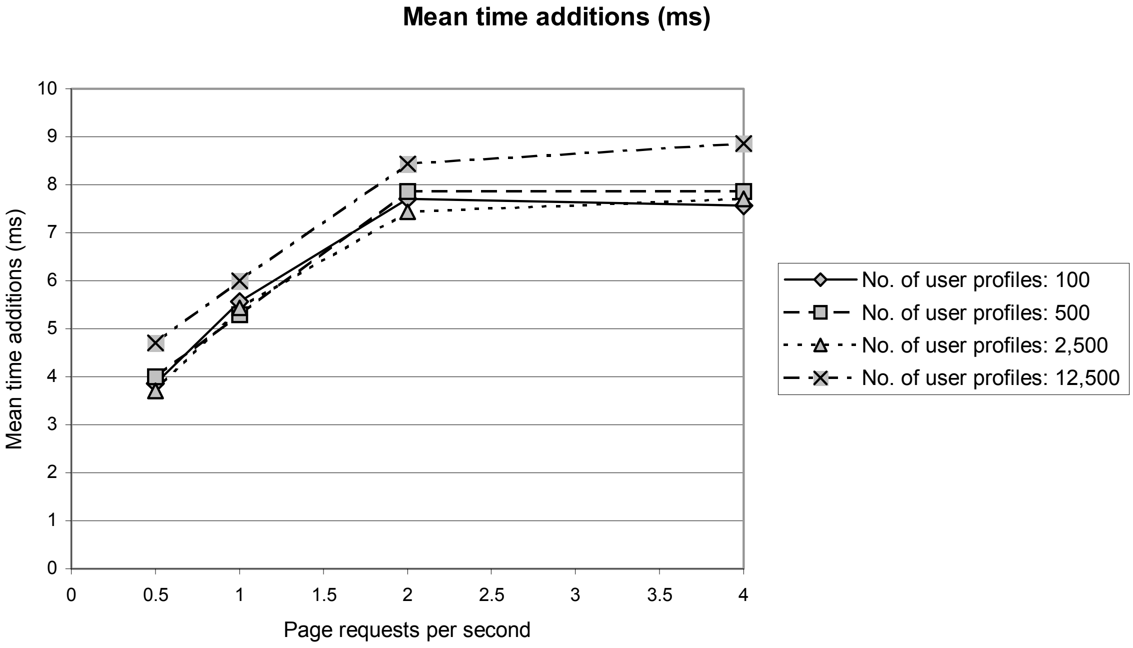


Figure 9-5: Mean time User Modeling Server add operations

Since all response times are smaller than 9 milliseconds, the add performance of our server can be regarded as superior. The excellent response time behavior is mainly due to a lean server configuration that takes advantage of a generously-sized database cache, several physical databases with optimized index structures for dedicated purposes (e.g., for maintaining event submissions), and the fact that recovery logging was disabled (see Chapter 9.2.4). Based on this streamlined configuration, our UMS can swiftly process many directory updates in cache memory.

Recovery logging becomes compulsory though in deployment scenarios where persistency is mandatory (see Chapter 2.2.4). Despite of increased efforts for recovery logging, we do not expect a significant drop in add performance (we found evidence for this in tests where recovery logging was enabled). Although directories are not optimized for update operations (see Chapter 4), their performance and scalability can be regarded as highly satisfactory for user modeling purposes.

9.3.1.2 Quality of Service

In this sub-chapter, we revisit our tests from the viewpoint of quality of service. The two most important technical factors that contribute to quality of service are the response times and response rates of our server from the perspective of client applications. As far as the response rate is concerned, no user model operation ever timed out or failed in any of our tests. Hence this factor seems optimal and we will not further look into it.

Table 9-3 analyzes the measured response times in more detail:

- The first two columns contain the independent parameters of our tests (namely the number of user profiles and page requests per second).
- The third column lists the mean response times for processing the four user model operations that personalize a Web page (namely three LDAP search and one add operations, see Chapter 9.1).
- The fourth and fifth column indicate for each test case the confidence interval for the means at the 99% level[116].
- The sixth column shows the means plus one standard deviation, and the last column the means plus two standard deviations.

From a usability point of view, our UMS exhibits an excellent response time behavior: all mean times plus one standard deviation are smaller than 78 milliseconds, and all means plus two standard deviations are smaller than 103 milliseconds (due to the small standard deviations and the huge sample size, the confidence intervals for the means are extremely narrow). This small user modeling overhead should enable client applications to stay well below the desirable response time limit of 1 second and, in any case, below the mandatory limit of 10 seconds. If we additionally take into account that no user model operation timed out or failed, we can conclude that the quality of service that our UMS offers can be regarded as highly satisfactory.

[116] Starting from the mean request times $\bar{x}$ and their standard deviations σ, we calculate the confidence interval for the mean response times of our user modeling server $\bar{x} \pm z\frac{\sigma}{\sqrt{n}}$. The central limit theorem and the fairly high number of page requests (our smallest n was 8,970) allows us to use a normal distribution for determining our z as 2.5762 (cf. Bleymüller et al. [1983]).

Performance results No. of user profiles and page requests/sec		Mean time page requests (ms)	Lower confidence limit mean time page requests (ms)	Upper confidence limit mean time page requests (ms)	Mean time page requests plus one standard deviation (ms)	Mean time page requests plus two standard deviations (ms)
100	**0.5**	11.15	11.04	11.26	15.27	19.38
	1	19.28	19.12	19.44	27.55	35.81
	2	30.84	30.64	31.04	45.34	59.84
	4	29.44	29.31	29.57	42.87	56.31
500	**0.5**	11.71	11.59	11.83	16.29	20.88
	1	17.29	17.15	17.43	24.62	31.95
	2	28.44	28.26	28.62	42.03	55.62
	4	27.15	27.03	27.27	39.65	52.14
2,500	**0.5**	11.00	10.88	11.12	15.58	20.17
	1	17.43	17.28	17.58	25.04	32.64
	2	29.30	29.12	29.48	42.87	56.43
	4	44.58	44.37	44.79	66.13	87.69
12,500	**0.5**	15.00	14.84	15.16	20.71	26.41
	1	25.71	25.50	25.92	36.73	47.75
	2	45.72	45.52	45.92	66.74	87.76
	4	52.57	52.33	52.81	77.49	102.42

Table 9-3: User Modeling Server quality of service black box perspective

Figure 9-6 analyzes the response time behavior for the four most challenging test cases in more detail. They all comprise 12,500 user profiles, and their workloads range from 0.5 to 4 page requests per second. In general, mean times and associated standard deviations increase only degressively with the workload. In all cases, mean response times plus one standard deviation are smaller than 78 milliseconds (and means plus two standard deviations are smaller than 103 milliseconds). Our simulated workload seemingly has not pushed the server to its performance limits.

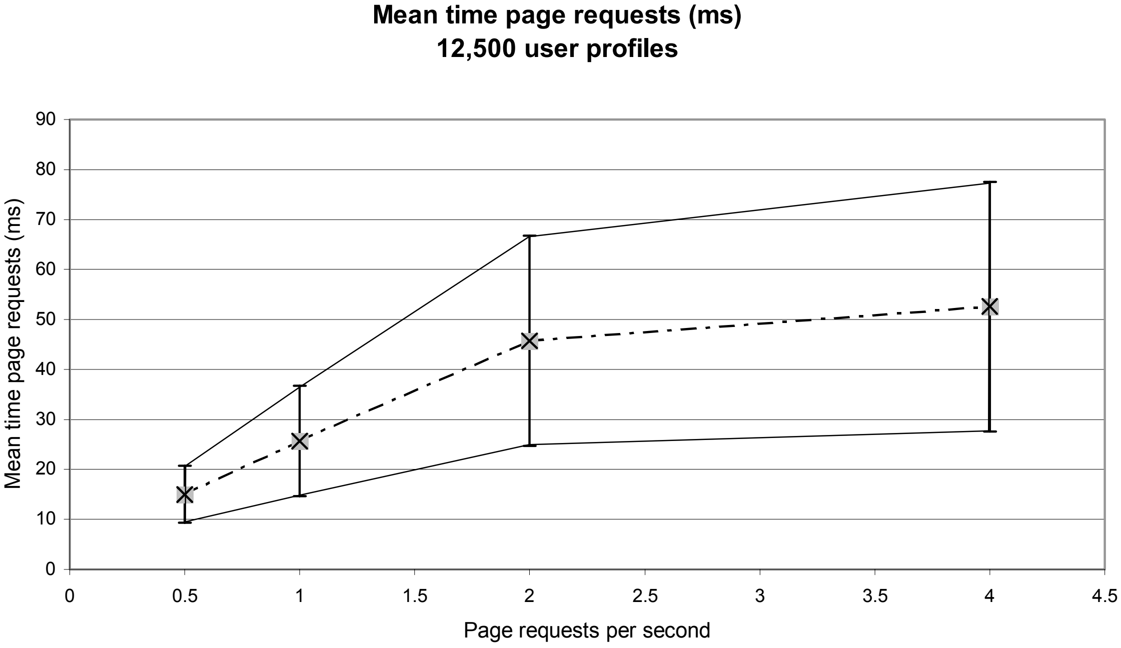


Figure 9-6: Mean time User Modeling Server page requests for 12,500 user profiles

9.3.1.3 Single Platform vs. Multi-Platform

In this sub-chapter, we briefly present the results from a deployment of the UMS to a multi-platform environment, and compare these results with the ones presented earlier for the single platform deployment scenario. The workload for this test was our most challenging one, i.e. 12,500 user profiles and 4 page requests per second. In the multi-platform scenario, only the Directory Component was running on the mentioned dual processor computer, and each learning component (i.e., ULC, MLC, and DIC) was deployed to a dedicated PC (for details regarding the hardware used, we refer to Chapter 9.2.4).

Figure 9-7 compares the resulting mean times and associated standard deviations on the levels of a Web application and of search and add operations. Each measure is shown for the single platform and the multi-platform scenario. Starting on the application level, we see that the mean time for processing the four user model operations that personalize a Web page plunges to 22.44 from 52.57 milliseconds (i.e., approximately 57% less), and its standard deviation to 10.54 from 24.92 milliseconds (i.e., approximately 58% less). The single most important reason for this improved performance is the considerably better search performance. The mean time needed for processing search operations falls to 5.29 from 14.57 milliseconds (i.e., approximately 64% less), and its standard deviation to 5 from 13.57 milliseconds (i.e., approximately 63% less). Less impressive is the performance gain for additions to the usage model: the mean time drops to 6.57 from 8.86 milliseconds (i.e., approximately 26% less), and its standard deviation to 6 from 8.29 milliseconds (i.e., 28% less).

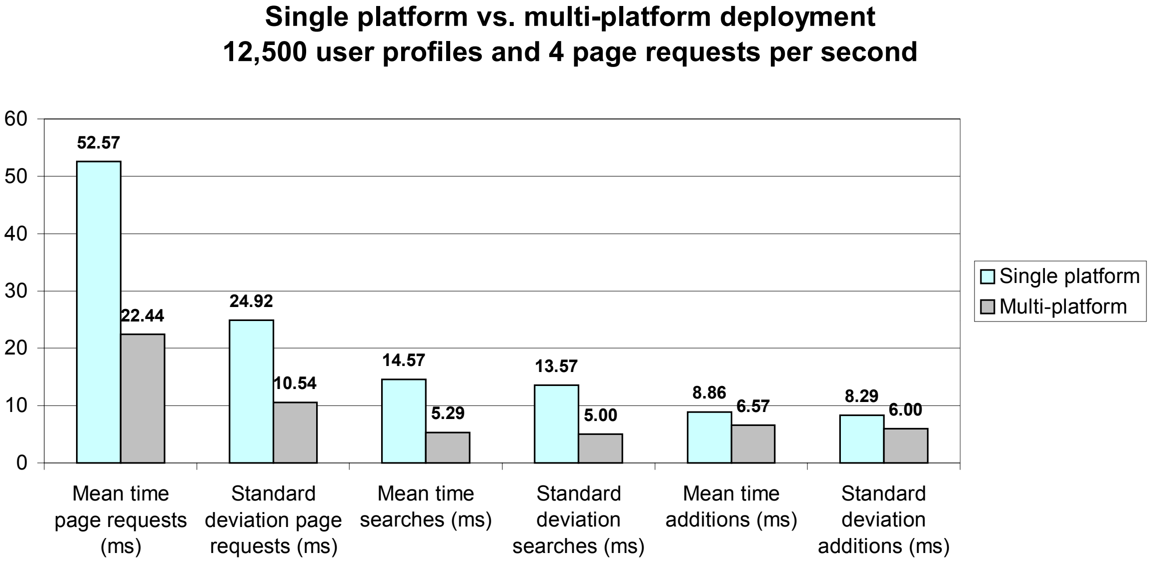


Figure 9-7: User Modeling Server single platform vs. multi-platform deployment

The distribution of our UMS across a network of four computers improved its performance considerably. In the multi-platform scenario, search performance benefits most from the relieved dual processor computer, since search operations can be concurrently processed by DS. Compared to that, additions with their inherent need for multi-user synchronization (see Chapter 2.2.1) can take less advantage of the additional hardware resources.

9.3.2 White Box Perspective

In this sub-chapter, we take a closer look at the User Modeling Components of the UMS, i.e. the ULC, the MLC, and the DIC. These components become triggered by LDAP update operations, and they operate concurrently to the Directory Component. On single-processor platforms, these components however compete with the Directory Component for hardware resources (especially processor time) and therefore impact the processing time for LDAP operations indirectly.

9.3.2.1 Performance and Scalability

ULC. The main purpose of the ULC is to acquire user's interests and preferences from features of objects a user has seen, rated, bought, etc. User-adaptive applications acquire these object features and communicate them as event vectors to the UMS. Within the server, the Scheduler forwards them to the ULC for further processing. The ULC maintains incoming event vectors in a dedicated queue and, as soon as a predefined number of vectors is available, starts processing them in a separate thread. After processing has been completed, the interest probabilities are persistently stored in the User Model. Due to the concurrent nature of the receiving and processing phase, the ULC can receive event vectors at any time. Other important advantages of this queue-based approach include (i) the graceful degradation of the ULC should it receive more events than it can process in a certain time interval and (ii) the avoidance of costly updates of the User

Model in secondary storage by maintaining interim learning results in main memory and persistently storing only the final interest probabilities in the User Model. For more information on the ULC, we refer to Chapter 8.4.

Figure 9-8 depicts the mean processing times of the ULC for acquiring users' interests and preferences. Mean times are shown for all 16 test cases.

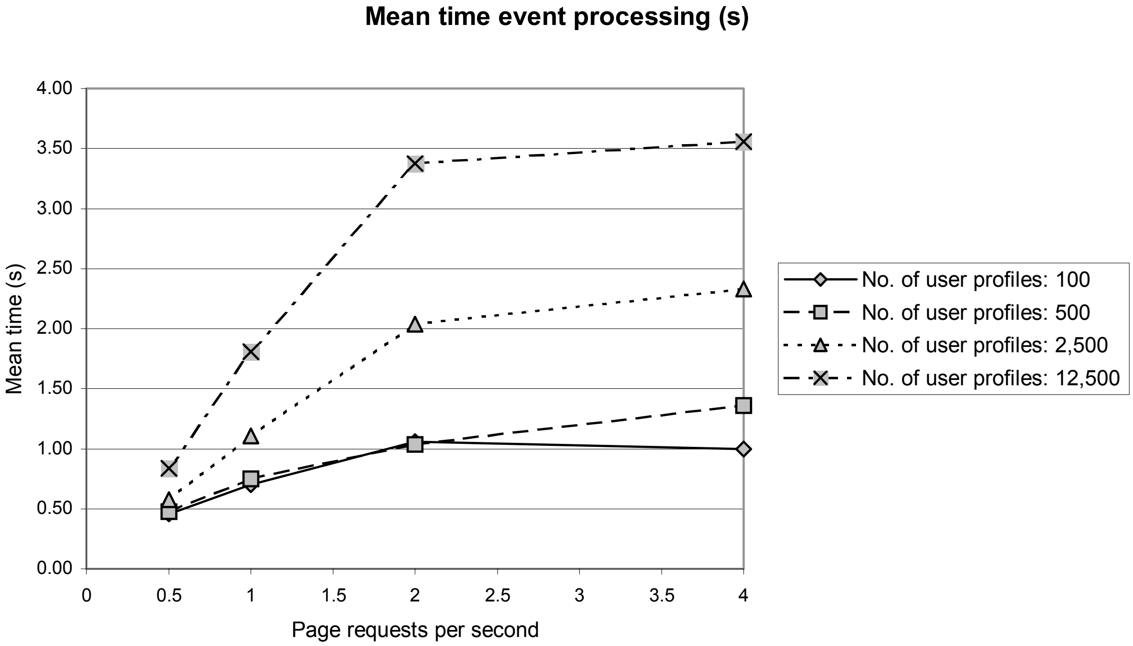


Figure 9-8: ULC mean time event processing

In general, mean times seem to mainly depend on the number of user profiles. They degressively grow with the number of page requests, which is mainly due to the mentioned queue-based architecture of the ULC (which allows for bulk processing of submitted events and for maintaining interim interest probabilities in main memory, thereby saving costly updates of the User Model).

Regarding performance, all recorded mean times are smaller than 4 seconds. This speed of learning can be regarded as highly satisfactory, since it permits keeping track of users' changing interests even between consecutive page requests. Interests that can be learned based on a page request are available to the Web application at the time of the subsequent page request, and can be used for adapting the latter page (for related requirements, we refer to Chapter 7.1). The ULC fully supports this 'inter-request' learning for all session types and test cases.

MLC. The MLC aims at predicting unknown characteristics in individual user profiles from known ones in profiles of a set of similar users. Assumptions about interests and preferences have either been provided by the users or were acquired by the ULC based on users' application system usage. The Scheduler, which is aware of all login operations to the UMS, forwards an event to the MLC for each user who logs into the system. Starting from this set of active users, the MLC searches for similar users in the User Model. As soon as the so-called neighbors are found, the MLC checks the profiles of active users for unknown interests and preferences and computes predictions for these 'holes' based on known interests and preferences in profiles of neighbors. In a final step, the MLC stores its predictions in active users' profiles. On an implementation level, the MLC employs a queue-based architecture like the ULC, which allows for the execution of queue management tasks concurrently with learning tasks. For more information on the MLC, we refer to Chapter 8.5.

Figure 9-9 shows the mean processing times of the MLC for predicting the interests and preferences of active users. Mean times are depicted for 13 of our 16 test cases.

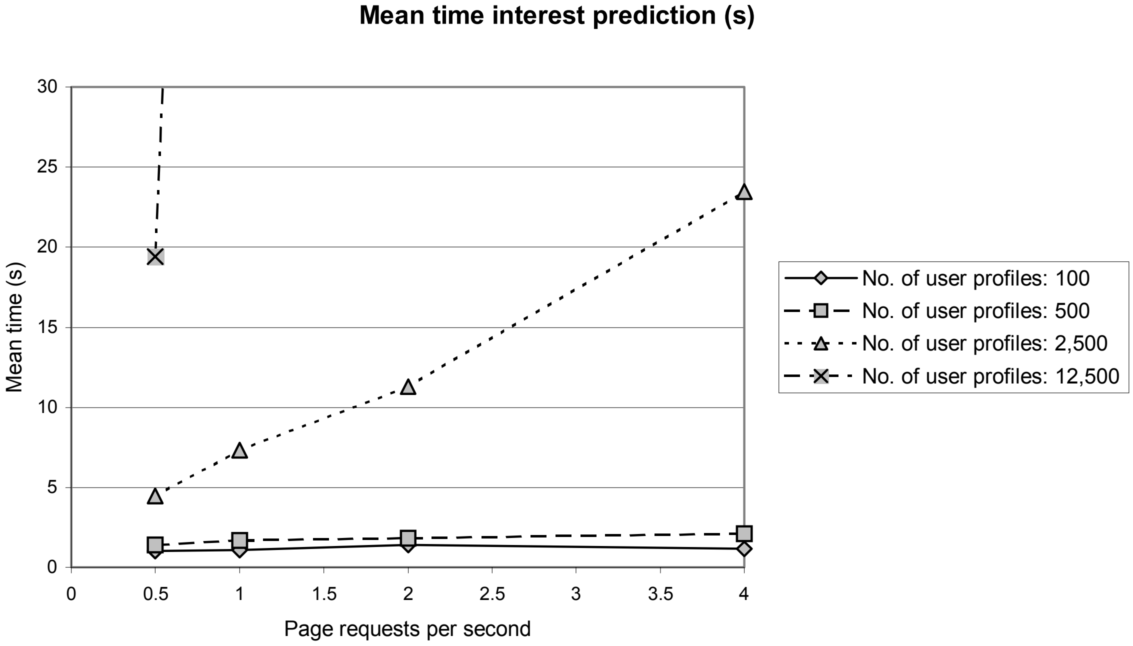


Figure 9-9: MLC mean time interest prediction (excerpt)

For 100, 500 and 2,500 user profiles, all mean times are smaller than 24 seconds, but show a progressive growth with increasing page requests. This performance can be regarded as satisfactory, since it allows for intra-session prediction of user interests and preferences for all seven session types and, except for Quickies, allows for the prediction of user interests and preferences even between consecutive page requests. Likewise acceptable are the 19 seconds mean time for 12,500 user profiles and 0.5 page requests per second.

In contrast, the mean processing time deteriorates considerably for 12,500 user profiles and higher numbers of page requests (which are no more depicted in Figure 9-9): 141 seconds for 1 page request per second, but 2:07:45 hours for 2 and 2:06:29 hours for 4 page requests per second. This is depicted in more detail in Figure 9-10, where the mean processing times of all 16 test cases are shown on a logarithmic scale. It seems that for 2 and 4 page requests per second, the MLC could not keep pace with the stream of user arrivals and approaches its performance limits. The convergence of the two mean times seems to support this assumption.

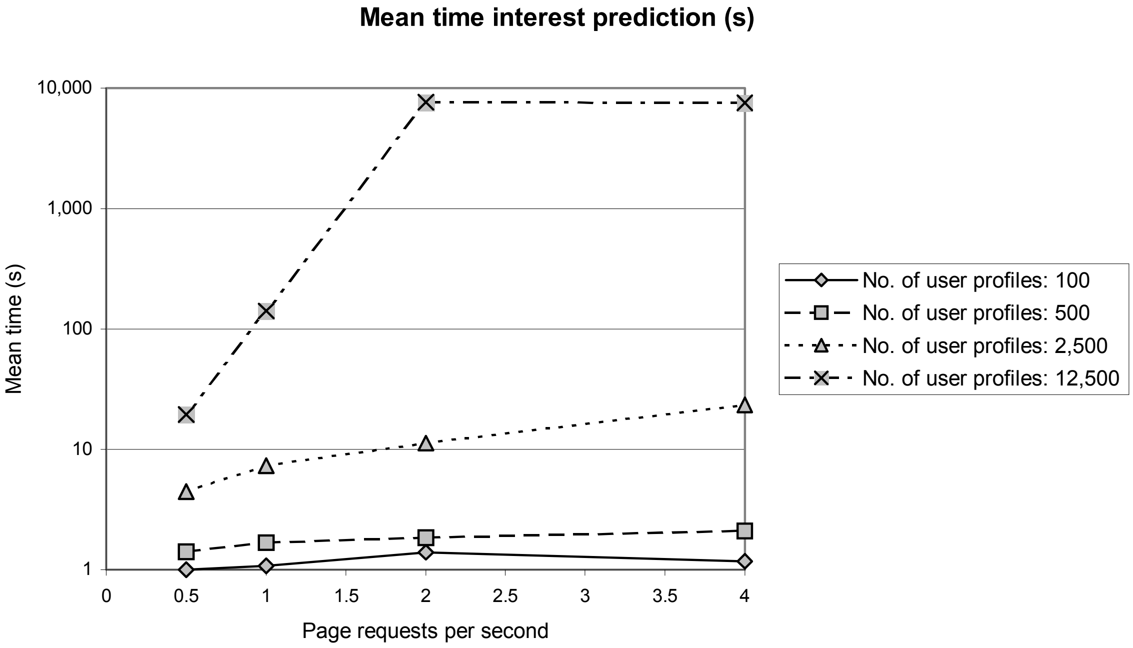


Figure 9-10: MLC mean time interest prediction

We believe that the performance and scalability of the MLC can still be regarded as satisfactory, though. This is due to the fact that the MLC we used in our tests searches similar users in the whole user population and does not restrict its search to a subset of user profiles, e.g. by applying statistical sampling methods. If we interpret the 100, 500, and 2,500 profiles used in our tests as samples from a larger set of user profiles, then the performance and scalability of the MLC seems again satisfactory. We highly recommend future work on the MLC that focuses on the employment of statistical sampling and/or singular value decomposition methods (see also our recommendation for future work in Chapter 8.5).

DIC. The main aim of the DIC is to quickly populate and update user models with inferred assumptions about users' interests and preferences. In order to achieve this, the DIC takes advantage of primary assumptions explicitly provided by users and implicitly acquired by the ULC and the MLC and propagates them upwards and sideways along the domain taxonomy.

From a technical point of view, the DIC registers with the Scheduler for additions, updates, and deletions in users' interest models. Subsequently, the DIC receives notifications from the Scheduler about relevant user model operations (e.g., the ULC inserts an assumption about a user's interest in `Umweltbelastung`, i.e. environmental burden), checks whether certain conditions apply (e.g., whether a minimum percentage of interests is available for sideways propagation), possibly draws domain inferences (e.g., the user is now assumed to be also interested in `Klima`, i.e. climate), and updates the user model accordingly. From an implementation point of view, the DIC relies on a queue-based architecture as the ULC and the MLC. For more information on the DIC, we refer to Chapter 8.6.

Figure 9-11 depicts the mean processing times of the DIC for inferring assumptions about users' interests and preferences. Mean times are shown for all 16 test cases.

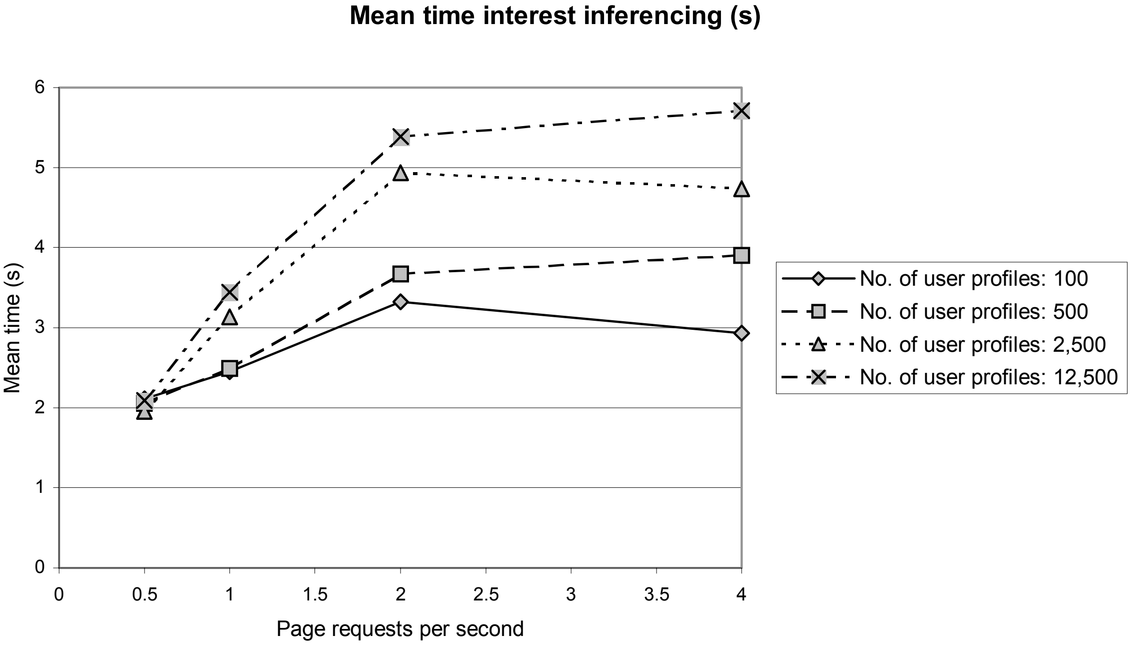


Figure 9-11: DIC mean time interest inferencing

The processing time behavior of the DIC resembles that of the ULC. Mean times depend on the number of user profiles and degressively grow (or even decrease in two cases) with the number of page requests. Like for the ULC, this behavior is mainly due to the queue-based architecture of the DIC.

All mean times that we collected are below 6 seconds. This learning speed can be regarded as highly satisfactory since it permits keeping track of users' changing interests even between consecutive page requests.

9.3.2.2 *Quality of Service*

We briefly present and discuss in the following the quality of service the User Modeling Components of our server exhibited during the tests. Following the definition we provided in Chapter 9.3.1.2, we define quality of service by the processing times and processing rates of the User Modeling Components from the perspective of the Directory Component (and indirectly also from the perspective of client applications). As far as the processing rate is concerned, no User Modeling Component ever timed out or failed in any of our tests; hence, we will not further look into this factor.

Table 9-4 analyzes the measured processing times in more detail:

- The first two columns contain the independent parameters of our tests (namely the number of user profiles and page requests per second).
- The third, fourth, and fifth column show for each component the upper limits of the confidence intervals for the mean processing times at the 99% level.

- The sixth, seventh, and eighth column list for each component the mean processing times plus one standard deviation, and the last three columns the means plus two standard deviations.

		Upper confidence limit mean processing time (sec)			**Mean processing time plus one standard deviation (sec)**			**Mean processing time plus two standard deviations (sec)**		
No. of user profiles and page requests/sec	**Components**	**ULC**	**MLC**	**DIC**	**ULC**	**MLC**	**DIC**	**ULC**	**MLC**	**DIC**
100	**0.5**	0.47	1.08	2.24	0.77	1.96	3.70	1.08	2.91	5.30
	1	0.71	1.15	2.57	1.19	2.03	4.20	1.69	2.97	5.95
	2	1.07	1.49	3.52	1.97	2.89	6.36	2.88	4.40	9.40
	4	1.01	1.26	3.09	1.84	2.82	5.99	2.68	4.47	9.04
500	**0.5**	0.48	1.47	2.16	0.77	2.27	3.40	1.07	3.13	4.75
	1	0.76	1.76	2.60	1.26	2.82	4.12	1.78	3.96	5.75
	2	1.04	1.92	3.82	1.68	3.34	6.40	2.33	4.83	9.12
	4	1.37	2.17	4.02	2.24	3.83	6.79	3.11	5.55	9.68
2,500	**0.5**	0.58	4.66	2.05	0.93	7.13	3.18	1.29	9.81	4.41
	1	1.12	7.65	3.27	1.86	12.81	5.45	2.62	18.28	7.77
	2	2.06	11.66	5.08	3.44	19.47	8.40	4.84	27.63	11.86
	4	2.34	24.30	4.85	3.90	50.15	8.14	5.47	76.85	11.53
12,500	**0.5**	0.86	20.42	2.21	1.57	32.46	3.60	2.30	45.49	5.10
	1	1.83	146.87	3.58	3.08	244.73	6.01	4.36	348.36	8.58
	2	3.39	7813.93	5.49	5.53	12729.10	9.04	7.68	17793.61	12.69
	4	3.58	7733.90	5.83	5.95	12577.07	9.70	8.35	17565.26	13.68

Table 9-4: User Modeling Server quality of service white box perspective

Against the background of the learning speed requirements we postulated in Chapter 7.1, the measured processing times of the ULC and the DIC seem very promising: all mean times plus one standard deviation are smaller than 10 seconds, and all means plus two standard deviations are smaller than 14 seconds (due to the small standard deviations and the huge sample size, the confidence intervals for the means are very narrow). This speed of learning permits keeping track of users' changing interests even between consecutive page requests for all session types and test cases. If we additionally take into account that no User Modeling Component ever timed out or failed in any of our tests, we can conclude that the quality of service that the ULC and the DIC offer can be regarded as highly satisfactory.

With regard to the MLC, the learning speed for 100 and 500 users seems satisfactory: all mean times plus one standard deviation are smaller than 4 seconds, and all means plus two standard deviations are smaller than 6 seconds. This performance allows for inter-request prediction of user interests and preferences for all seven session types. For 2,500 users, the learning speed of the MLC is less satisfactory: mean times plus one standard deviation are smaller than 51 seconds, and means plus two standard deviations are smaller than 77 seconds. This performance still allows for inter-request prediction of user interests and preferences for Single Mission, Do It Again, and Loitering sessions. Intra-session personalization is additionally supported for Information Please and Surfing sessions.

For 12,500 users, however, the learning speed of the MLC deteriorates considerably: the mean processing time plus one standard deviation rises from 32.46 seconds for 0.5 page requests per second to approximately 3:32:09 hours for 2 and the mean plus two standard deviations from 45.49 seconds for 0.5 to approximately 4:56:33 hours for 2 page requests per second. This unacceptable performance underlines our recommendation for future work on the MLC we made in Chapter 8.5.

10 Discussion

In this chapter, we revisit the requirements catalogs that we worked out in Chapters 2 and 3 and discuss our user modeling server against this background. We organize our discussion according to the structure we introduced earlier, i.e.

i. *server-related* requirements and

ii. *user modeling-related* requirements.

In order to facilitate orientation, we list the requirements as run-in headings in italics.

10.1 Server Requirements

Multi-user synchronization. In Chapter 2 we argued that a concurrent server design can be regarded as highly suitable for user modeling servers. Concurrent servers maintain several threads for processing requests, and control the amount of server resources that they utilize for processing requests. Our user modeling server adheres to these principles. On the component level, the Scheduler mediates between the concurrently operating components of our server. And within each component, several concurrent processing threads are maintained for communication and learning tasks. The Directory Component of our server uses a configurable number of threads for processing incoming requests. In the experiments we described in Chapter 9, we used sixty concurrent threads for processing incoming requests. Likewise, each of our learning components maintains a configurable number of processing threads. The results of the experiments in Chapter 9.3 demonstrate the positive effect of our design decisions on the performance and scalability of our user modeling server.

Transaction management. The transaction management offered by our user modeling server is relatively poor in comparison to current database and transaction management systems (see Chapter 2). Our server provides transactional semantics (i.e., ACID properties) for update operations only. Each update operation is an indivisible unit of work, which either modifies all or none of the attributes of an entry to which it is applied (Atomicity). The compliance of the User Model with the directory schema is strictly preserved (Consistency), and concurrently executed LDAP operations do not affect update operations (Isolation). Only successful updates are made persistent (Durability) and become visible for subsequent operations.

When comparing the transaction management of our user modeling server with those of the PAT-InterBook system (the only academic user modeling system of which we are aware that provides transaction management [Brusilovsky et al., 1997]), we find several advantages as well as disadvantages (see Chapter 2). Advantages of our approach include that (i) all users of user-related information (e.g., administrative users, ‘real’ users, applications, components of the user modeling system) can take advantage of our facilities and that (ii) full transactional support (i.e., Atomicity, Consistency, Isolation, and Durability) is provided for update operations. A disadvantage of our current design is that the scope of a consistency context is restricted to a single update operation (see Chapter 5.3.2).

The basic transactional facilities of our user modeling server seem adequate for user-adaptive applications with rather moderate consistency requirements (e.g. WebGuide). If needed, however, these transactional facilities can be easily matured, e.g. by employing the respective facilities of a database management system. In this case, the interface of our user modeling server has to be extended by custom user model operations that allow for communicating consistency contexts. Examples of such operations are ‘BeginOfTransaction’, ‘EndOfTransaction’, and ‘RollbackTransaction’. A consistency context is communicated by BeginOfTransaction and EndOfTransaction, whereas the rollback of a user model to the state before the transaction started is communicated by RollbackTransaction. From an implementation point of view, these custom operations can be implemented using the extended operation facility of DS (see Chapter 5.3). The standard entry/fetch plug-in of DS for accessing the default database management system has to be replaced by a custom one that interfaces to an appropriate database management system. Finally, the aforementioned custom operations for communicating consistency contexts have to be connected with the custom entry/fetch plug-in (see Chapter 8.3).

Query and manipulation language. There is considerable evidence in the literature that recommends LDAP as a query and manipulation language for a broad range of applications (e.g., Howes et al. [1999], Shukla and Deshpande [2000]). Clients can access our server via a variety of LDAP SDKs (see Chapter 8). Custom operations that facilitate user model management (e.g., for creating and deleting user models) relieve administrators and application programmers from laborious and error-prone administration and programming tasks (see Chapter 8.3).

The client interface of our user modeling server caters quite well to the requirements that we introduced in Chapter 2. Due to LDAP's design for a broad range of applications and its compliance with many standards (see Chapter 4), our client interface can be assumed to meet the requirements of application and domain independence very well. The functional model of LDAP (see Chapter 5.3) comprises a scalable set of generic language elements (e.g., search, add, delete, rename, modify) with comparison operators (e.g., presence, equality, greater, approximate) and logical operators (and, or, not). Operators from relational calculi (e.g., union, intersection) are however not supported. LDAP allows for a descriptive specification of user model operations in a set-oriented manner (see Chapters 5.3 and 8.2). The LDAP client APIs are available for a variety of languages (e.g., C/C++, Java, Perl) and operating systems (e.g., Windows NT, Linux, Solaris). Most of these APIs can be assumed to support computational closure. The API for the C programming language (as defined in RFC 1487 [Howes et al., 1995]), e.g., encapsulates results from LDAP operations in a common data structure called `LDAPMessage`. This structure can be filled by, e.g., a search operation, and subsequently be used in an add operation without any modifications. With regard to exhaustiveness, LDAP version 2 had several shortcomings, most notably its lack of client-side schema retrieval and manipulation features. In version 3 (see RFC 2252 [Wahl et al., 1997b]), however, these features are included; hence, we can assume the requirement exhaustiveness being covered as well. Regarding performance and quality of service, the results of our experiments propose that LDAP caters to these requirements very well (see Chapter 9.3). And regarding security, the LDAP SDKs from Netscape [2000b] we used provide sophisticated facilities for authenticating clients and servers and for encrypting the flow of information between them (see Chapter 5.4). The only requirement that is not supported by our client interface is a layered architecture. We assume that such an architectural characteristic has been dispensable due to the predominance of TCP/IP as a communication protocol and LDAP's extended operation feature successfully covering requirements for custom extensions.

Persistency. If the persistency feature of our user modeling server is enabled, all information hosted by our server can be assumed to survive even unpredictable events such as system breakdowns and hard disk crashes. In order to achieve this, our server takes advantage of respective Directory Server facilities, which record each internal database operation in a database transaction log (for more information, we refer to iPlanet [2000b]). If the server were to experience a failure, such as power outage and abnormal shut down, it automatically detects the error condition after its restart and uses the database transaction log for recovering its database. After recovery has been completed, clients can continue accessing the server, ideally without any loss or damage of information[117].

The support for persistency our server offers can be characterized as (i) static, since the respective facilities of DS need to be enabled before starting the server, and (ii) (very) coarse-grained, since persistency can only be associated with all models hosted by our server, i.e. without providing any further means for differentiating this association (e.g., only for Usage Model contents).

Integrity. Based on LDAP's information and naming model (see Chapters 5.1 and 5.2) and on complementary features provided by DS, our user modeling server can preserve

[117] One precondition for a successful recovery is that the database transaction log is not corrupted.

integrity on different levels of granularity. On the level of single assumptions, the object classes of an LDAP entry specify the set of applicable mandatory and optional attributes. Each attribute is associated with an attribute type, which denotes its representation format (e.g., `Integer`, `DirectoryString`) and related operations (e.g., `integerMatch`). In Chapter 5.1, we exemplified this for the attribute type `description`, which restricts all attribute values to a `DirectoryString` with a maximum length of 1,024 characters and associates them the matching rules `caseIgnoreMatch` for equality and `caseIgnoreSubstringsMatch` for sub-string matching.

For preserving referential integrity between related assumptions, DS can automatically reflect updates to related entries. If a user removes her User Model from the directory, e.g., then related entries in groups the user belongs to are also removed by the server. If the referential integrity facility is not enabled, these related entries would remain in the directory tree. In order to preserve referential integrity, DS records each delete or modify operation to the referential integrity log. After a configurable amount of time, the server processes this log as follows:

- if an entry was removed, then DS also removes related entries;
- if an entry's attributes have been changed, then DS changes attributes of related entries accordingly.

By default, referential integrity is enabled and preserves integrity for the `member`, `uniquemember`, `owner`, and `seeAlso` attributes *immediately* after a delete or modify operation is carried out. Administrators can remove attributes from or add new attributes to this list.

For preserving custom integrity constraints between related assumptions, DS provides programmers the following interfaces for hooking custom functionality into its processing loop (see Chapter 8.3):

- *Pre-operation*: DS calls these plug-ins *before* performing an LDAP operation. This allows programmers to apply custom integrity constraints (e.g., whether the presumable domain expertise of a user changes either from 'beginner' to 'intermediate' or from 'intermediate' to 'expert') and, depending on the outcome of these checks, prevent the server from (e.g., reject an update from 'beginner' to 'expert') or allow the server to execute an LDAP operation (e.g., permit an update from 'beginner' to 'intermediate').
- *Post-operation*: DS calls these plug-ins *after* performing an LDAP operation. This allows programmers to preserve integrity after an LDAP operation has been executed (e.g., since a user's presumable expertise in one domain changed from 'intermediate' to 'expert', her presumable expertise in a closely related domain has to be modified accordingly).

These pre- and post-operation facilities are not limited to integrity preservation, but can be used for implementing a rather broad range of custom tasks (e.g., integrating a database management system, encrypting and decrypting directory content, implementing extended operations, implementing extensible matching filters).

Access control. The access control facilities of our user modeling server can be regarded highly suitable for user modeling purposes. As already discussed in Chapter 5.4, the respective mechanisms of our server rely on the fine-grained access control model provided by DS. It is based on access control lists and allows for implementing a broad range of access control models. Together with the sophisticated security and distribution mechanisms of our user modeling server, these facilities can be assumed to cater to a wide range of security and privacy requirements (see Chapter 6.3.3).

10.2 User Modeling Requirements

Functionality and input data. Many specific requirements that guided the design of our UMS stem from the Deep Map project (see Chapter 7.1). Today, WebGuide is the only Deep Map component that takes advantage of our user modeling server. In the near future, however, the number of clients can be assumed to significantly increase [EML, 2000]. The range of potential input data that our user modeling server can manage comprises information about the user (i.e., demographic characteristics, interests and preferences), the system's usage (e.g., navigational behavior), and the usage environment (e.g., hardware and software environment). Apart from that, LDAP's versatility regarding input data and the high degree of directory standardization (e.g., RFC 2251 [Wahl et al., 1997a], RFC 2252 [Wahl et al., 1997b], RFC 2256 [Wahl, 1997], and related X.500 standards like X.520 [ITU-T, 2001c] and X.521 [ITU-T, 2001d]) increase the range of potential input data even further. In this regard, our user modeling server seems to clearly excel the commercial user modeling servers we presented in Chapter 3.2 (e.g., GroupLens, Personalization Server).

Acquisition methods and representation. Our user modeling server for Deep Map implements three complementary acquisition methods. The ULC acquires users' interests and preferences from features of objects users have seen, rated, bought, etc. The MLC predicts unknown characteristics in users' profiles from sets of profiles that are similar with regard to the known characteristics. And the DIC applies domain inferences to individual user models, thereby propagating interests in users' profiles sideways and upwards along the domain taxonomy. As already pointed out earlier, the combination of these acquisition methods leverages several synergistic effects between the employed learning techniques and compensates for well-known deficits of individual techniques with regard to, e.g., performance, scalability, integration of domain knowledge, sparsity of data, and cold start. For more information, we refer to Chapters 8.4, 8.5, and 8.6.

In the following, we briefly discuss these acquisition methods along the deployment requirements we presented in Chapter 3:

- *Scope of applicability*. Our acquisition methods seem to cover a wide range of user modeling tasks (e.g., acquisition of users' interests and preferences from usage data, prediction of users' demographic characteristics from models of similar users, inference of interests and preferences by applying domain inferences). The support for user modeling tasks that our server provides goes far beyond that of the commercial user modeling servers we discussed in Chapter 3.2 (e.g., GroupLens, Personalization Server). The same can be anticipated for the range of input data types.
- *Facility for incremental learning*. The acquisition methods provided by our server support incremental learning. The user-related information maintained by our server can keep pace with a user's changing interests and preferences, which is a main motivation for personalization from a marketing point of view (see Chapters 1.2 and 9.3). Quite comparable to commercial systems like FrontMind (see Chapters 3.2.3 and 3.3), our server exhibits both deterministic and non-deterministic (i.e., probability and similarity based) personalization behavior. Deterministic behavior is exhibited by the

domain inferences of the DIC, and non-deterministic behavior is exhibited by the usage-based learning of the ULC and the collaborative filtering of the MLC.

- *Explicitly represented knowledge*. Our user modeling server takes advantage of a widely used information repository for explicitly representing its models. In this regard, our approach is comparable to the one taken by commercial systems like Personalization Server and FrontMind[118]. Whereas Personalization Server and FrontMind rely on relational database management systems (e.g., from vendors like Microsoft and Oracle) for storing (most of) their models, our server takes advantage of an LDAP-based directory. We believe that our decision for LDAP provides significant advantages with regard to extensibility, management of distributed information, replication scale, performance and scalability, and compliance with standards (for more information, we refer to Chapter 4). Our choice of an LDAP-based directory also facilitates the access to user-related information for user-adaptive applications via standard interfaces like LDAP and ODBC. This openness clearly contrasts with the proprietary models hosted by systems like GroupLens, which can only be accessed via custom APIs.
- *Employing domain knowledge in the learning process*. In our server, domain knowledge is represented in the domain taxonomy and can be employed for guiding learning processes. As a part of the System Model, the domain taxonomy comprises assumptions about users' demographic characteristics and their interests and preferences (see Chapter 8.2.3). Apart from the domain taxonomy, the System Model also contains information that controls learning. Examples include `Classifiers`, which control the discretization of continous attribute values (e.g., age, income) and attributes like `mentor_prediction` and `mentor_finding`, which control whether predictions are computed from a set of similar users and whether specific attributes are included in the process of finding mentors, respectively. Compared to Learn Sesame's MDL facilities that we discussed in Chapters 3.2.4 and 3.3, however, our domain modeling mechanisms seem less generic, although with the benefit of reduced administrative burden.

Extensibility. The open architecture of our user modeling server allows custom user modeling functionality 'to be plugged' into our server at any time. Examples of such functionality include the identification of recurrent patterns in usage data (cf. the learning scenario we described for Learn Sesame in Chapter 3.2.4) and the computation of models of user groups from similar characteristics in individual user models (e.g., by applying Bayesian clustering [Orwant, 1995; Paliouras et al., 1999]). In order to support the integration of custom functionality into our server, we put stub components at the disposal of developers that simplify the communication with our Directory Component. Additional tasks that have to be accomplished are the registration of event subscriptions with the Scheduler (see Chapter 8.3) and the creation of accounts for the new learning components (see Chapter 8.2.4). We assume that all these tasks can be easily accomplished by an experienced programmer.

Besides this rather tight integration of custom user modeling functionality into our server, quite a few complementary software products can be loosely integrated via their LDAP and

[118] FrontMind relies on proprietary files for maintaining models of users' behavior.

ODBC interfaces (see Chapter 8). Throughout this thesis project, we took advantage of these facilities several times, e.g. (i) for administering user model contents via an LDAP editor/browser (see Chapter 8.2), (ii) for visualizing and analyzing user model contents via InfoZoom, and (iii) for integrating Directory Mark into our Test Bed (see Chapter 9.2.1). This seems to verify the statement we made in Chapter 3.3 that a user modeling server with open interfaces for complementary tools can successfully cope with the broad range of requirements for a personalization platform.

Integration of external user and usage information. The distribution, replication, and meta-directory facilities of our user modeling server allow for the integration of user-related information into a (virtually) centralized user model (e.g., client profiles from ERP systems, client segmentations from database marketing). From the viewpoints of a client and an application programmer, this enables common applications (e.g., WWW browsers, e-mail clients) and tools (e.g., directory browsers) to transparently access user-related information and associated user modeling services. We introduced the integration facilities of our server in Chapter 4 and demonstrated their support for three advanced user modeling scenarios in Chapter 6.3. Together with the access facilities to user model contents we presented earlier, we believe that this (i) allows organizations to leverage the assets of user-related information (see Chapter 1.2) and (ii) allows for a more holistic personalization, both from the viewpoint of organizations and users (see Chapter 1.3). In this vein, the integration and access facilities of our server seem to excel even those provided by commercial servers like Personalization Server and FrontMind (see Chapter 3.2).

Privacy. Our user modeling server provides support for authentication, signing, encryption, access control, auditing, and resource control. We introduced these features in Chapter 5.4, and demonstrated their use for safeguarding security and privacy in a scenario presented in Chapter 6.3.3. Based on these features, user model developers should be able to design and implement appropriate security and privacy policies.

Architecture. The user modeling server we developed relies on a component-based multi-tier architecture (see Chapters 6, 7, and 8). The main advantages of this architecture include performance, scalability, and flexibility. In our experiments, we empirically verified that our server can fully cope with small and medium-sized application workloads (see Chapter 9). The processing time for a representative mix of user modeling operations was found to only degressively increase with the frequency of page requests. The distribution of the user modeling server across a network of computers considerably improved its performance.

The granularity of distribution and the associated scalability surpasses that of GroupLens, Personalization Server, and Learn Sesame. In all these systems, the ‘critical’ user modeling functionality is incorporated into a single tier. The employment of differently configured instances of user modeling components is likewise not supported by these commercial user modeling servers and, to the best of our knowledge, not planned to be available in the near future. We already presented an example that motivates this requirement in Chapter 3.3, i.e. dedicated learning strategies that can be accomplished by using differently configured instances of the same learning engine.

Regarding the architectural requirements we discussed in Chapter 3.3, we conclude that our user modeling server allows for (i) the flexible distribution of components across a network of computers, according to resource and availability requirements, (ii) the employment of differently configured instances of user modeling components, and (iii) the integration of complementary software products.

Software and hardware. We already mentioned in Chapter 7 that our user modeling server can be accessed via LDAP and ODBC. Programmers can accomplish this by taking advantage of a variety of SDKs and interface components, e.g. the LDAP C SDK from the University of Michigan, various Directory SDKs from Netscape, built-in LDAP support in Java from Sun and Microsoft, JNDI from Sun, and ADSI from Microsoft. These SDKs and interface components integrate with common component frameworks (e.g., Active X, COM) and programming languages (e.g., C/C++, Java, Basic, Perl). As of the time of writing, the number of e-commerce servers (e.g., BroadVision's One-To-One, Vignette's StoryServer), (generic) Web development environments (e.g., Allaire's ColdFusion, Apple's WebObjects [Apple, 2000]), and ERP systems (e.g., SAP [2001], Baan [2001]) that support LDAP is rapidly increasing. Truog et al. [1999] predicts that all major systems will support LDAP soon. Compared to the access facilities provided by the commercial user modeling servers we reviewed in Chapter 3.2, this support seems quite comprehensive.

The current version of our user modeling server relies on iPlanet's Directory Server and runs on Windows NT and Solaris. Porting our user modeling server to other directory servers (e.g., Active Directory Server) and operating systems (e.g., Linux, AIX) can be assumed a rather easy task, since most of our server components are written in Java (except for the Scheduler, which is written in C/C++) and communication between components is established via CORBA and LDAP.

11 Summary and Perspectives

The user modeling server that we developed in this thesis is an open, standards-based, and platform-independent tool that provides essential user modeling services to user-adaptive applications. While previous user modeling systems stored data about users in database and knowledge representation systems, our server employs a directory management system for this purpose. This offers significant advantages with respect to the

- management and retrieval of (user-related) information, in a way that is compliant with established standards;
- definition of new (user-related) information types;
- distribution of information across a network, which often leads to better performance, scalability, availability, and reliability of the user modeling service;
- replication of information, which may enhance the performance and availability of the overall service, and is particularly useful in mobile applications where clients can become disconnected from the network; and the
- security of information and users' privacy, by providing facilities for authentication, signing, encryption, access control, auditing, and resource control.

Applications can take advantage of a set of core techniques for drawing assumptions about users. By integrating these techniques into a single server, synergistic effects between them can be leveraged, thereby compensating for shortcomings of individual techniques (e.g., in case of performance or scalability problems, or when data is sparse or not yet available). New techniques can be easily 'plugged into' the server at any time.

We argued that our user modeling server provides at least basic support for the rather broad range of requirements that we collected. Our empirical evaluation verified that our server can fully cope with the workloads of small and medium-sized application environments. We found that the processing time for a representative real-world mix of user modeling operations only degressively increases with the frequency of page requests. The distribution of the user modeling server across a network of computers additionally improved its performance. At the same time, the hardware demands of our server are moderate.

More recent experience that we gained from deploying our user modeling server to large commercial Web sites suggests that our server can be deployed to high-workload environments as well. The most notable large-scale experiment that we conducted comprised 8 million user profiles[119] and a workload of approximately 42 Web page requests per second[120]. In order to realize this workload, we employed a total of 1,794 simultaneous Clients in several Test Beds. The UMS was installed on a 'Fire V880' from Sun's entry-level server segment [Sun, 2002b]. This computer was equipped with eight 750 MHz processors, 8 MB cache per processor, 32 GB of RAM, and more than 200 GB disk space. The software used was Solaris 8 and iPlanet Directory Server 5.1. In order to take full advantage of the generous hardware resources available, we increased the cache settings for the Directory Component and each of our learning components to 2 GB. As for the rest, the design of this experiment was very comparable to the one we described in Chapters 9.1 and 9.2.

The results were very encouraging. From the black box perspective of a client application, our UMS showed a mean response time of 35 ms for personalizing a Web page (i.e., for processing three LDAP search and one add operations). This user modeling performance should easily allow a personalized application to stay well below the desirable response time limit of 1 second and, in any case, below the mandatory limit of 10 seconds. None of the several million search and add operations that were submitted by our simulated users failed or timed out.

Another lesson we learned from deploying our server to real-world environments was in terms of hardware sizing. We found that the sizing characteristics of our server closely resemble those reported in the literature for its Directory Component. For example, Nelson [2002] mentions the following rules of thumb for the number of CPUs that are necessary for processing LDAP operations: "With Directory Server 4.0, search performance will scale almost linearly with the addition of up to 4 CPUs. In this range, you can expect to see 500-1,000 queries per second for each CPU. Beyond 4 CPUs, the resulting increase in performance per CPU is less but still significant".

The resource needs of our User Modeling Components depend on the number of these components (each can be present or absent, and instantiated multiple times) as well as on several parameters that determine, e.g., the learning frequency, the size of the correlation space, etc. As far as the allocation of processor resources is concerned, we found that an even distribution between the Directory Component and the User Modeling Components is a good approximation. A constant amount of main memory is needed for hosting the

[119] MSN had about 8 and AOL 34 million subscribers at the end of July 2002 [Jupitermedia, 2002].

[120] This workload roughly equals that of the largest German news portal with nearly 15 million unique users [NetRatings, 2002], which is about 15-20% the size of the top three U.S. portals.

components of our user modeling server and the operating system. Variable amounts of memory are required for the various caches that are maintained by Directory Server and our User Modeling Components. The optimal sizes of these caches depend mainly on the number of user profiles, their mean number of entries, the mean size of a user profile entry, the access frequency of individual user profile entries, and the evenness of users' login into the user modeling server. Similar considerations apply to the required amount of disk space. Nelson [2002] gives further advice on the estimation of these values.

We believe that our work impacts the design, implementation, and deployment of user modeling and user-adaptive systems both in research and commercial environments. Our user modeling server has been already successfully deployed to commercial application environments with several millions of users. We regard our empirically founded approach of simulating the user modeling workload of real-world application environments as highly promising. It allows us to experimentally verify and predict the deployment characteristics of a user modeling server under various workload conditions. Our experience with actual installations of our server in commercial environments confirmed that this approach and the developed simulation test bed were an indispensable tool for real-world personalization.

Throughout this thesis, we pointed out several avenues for future work that we deem worthwhile to investigate. Especially promising are refinements of the learning techniques we used in our User Modeling Components (especially in the MLC) and the employment of complementary learning techniques.

Bibliography

1to1Web (2000). One-to-One Web Marketing. C. Allen, D. Kania and B. Yaeckel. http://www.1to1web.com

Åberg, J. and Shahmehri, N. (1999). Web Assistants: Towards an Intelligent and Personal Web Shop. In Proceedings of *Second Workshop on Adaptive Systems and User Modeling on the World Wide Web at WWW-8 and UM99*. Toronto (Canada) and Banff (Canada), 5-12. http://www.contrib.andrew.cmu.edu/~plb/WWWUM99_workshop/aberg/aberg.html

About.com (2000). Artificial Intelligence. About.com. http://www.ai.about.com

Abrams, C., Bernstein, M., deSisto, R., Drobik, A. and Herschel, G. (1999). E-Business: The Business Tsunami. In Proceedings of *Gartner Group Symposium/ITxpo*. Cannes (France).

Accrue (2000). Accrue Software. http://www.accrue.com

Allaire (2000). ColdFusion. Allaire.
23. July 2000. http://www.allaire.com/Products/ColdFusion
30. September 2003. http://www.macromedia.com/software/coldfusion/

Allaire, J. and Bestavros, A. (2000). Advanced Profiling and Personalization Strategies. *Personalization Summit*, San Francisco (CA).

Allen, C., Kania, D. and Yaeckel, B. (1998). *Internet World Guide to One-To-One Web Marketing*. New York (NY): John Wiley and Sons.

Almeida, V., Bestavros, A., Crovella, M. and Oliveira, A. (1996). Characterizing Reference Locality in the WWW. In Proceedings of the *Fourth International Conference on Parallel and Distributed Information Systems*. IEEE Computer Society, 92-103.

Alspector, J., Kolcz, A. and Karunanithi, N. (1997). Feature-based and Clique-based User Models for Movie Selection: a Comparative Study. *User Modeling and User-Adapted Interaction* 7(4): 279-304.

Amazon (2000). Amazon.com Privacy Notice. Amazon.com. http://www.amazon.com/exec/obidos/subst/misc/policy/privacy.html

Ambrosini, L., Cirillo, V. and Micarelli, A. (1997). A Hybrid Architecture for User-Adapted Information Filtering on the World Wide Web. In A. Jameson, C. Paris and C. Tasso, eds., *User Modeling: Proceedings of the Sixth International Conference*. Wien, New York (NY): Springer, 59-61. http://www.cs.uni-sb.de/UM97/gz/AmbrosiniL.ps.gz

Andrews, G. (1991). Paradigms for Process Interaction in Distributed Programs. *ACM Computing Surveys* 23(1): 49-90.

Angoss (2000). Angoss Software. http://www.angoss.com

Appian (2000a). Appian. http://www.appiancorp.com

Appian (2000b). Appian Web Personalization Report. Appian.
9. January 2000. http://www.appiancorp.com/awpr.asp

Apple (2000). WebObjects. Apple. http://www.apple.com/webobjects

Ardissono, L. and Goy, A. (1999). Tailoring the Interaction with Users in Electronic Shops. In J. Kay, ed., *UM99 User Modeling: Proceedings of the Seventh International Conference*. Wien, New York (NY): Springer, 35-44.

Ardissono, L., Goy, A., Meo, R. and Petrone, G. (1999). A Configurable System for the Construction of Adaptive Virtual Stores. *World Wide Web* 2(3): 143-159.

Ardissono, L. and Goy, A. (2000). Tailoring the Interaction with Users in Web Stores. *User Modeling and User-Adapted Interaction* 10(4): 251-303.

Arlitt, M. and Williamson, C. (1996). Web Server Workload Characterization: The Search for Invariants. In Proceedings of the *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. New York (NY): ACM, 126-137.

ASOC (2000). sphinxVision – Knowledge Processing Tool. ASOC.
23. January 2000. http://www.asoc.com/e_home.html

ATG (2000). ATG Products. Art Technology Group (ATG).
http://www.atg.com/products

Autonomy (2000). Autonomy Systems. http://www.autonomy.com

Baan (2001). Baan.com. http://www.baan.com

Bachem, C. (1999). Profilgestütztes Online Marketing. In S. Tjoa, ed., *Personalisierung im E-Commerce*. Hamburg (Germany), Section 13.

Balabanovic, M. (1997). An Adaptive Web Page Recommendation Service. In Proceedings of the *1st International Conference on Autonomous Agents*. Marina del Rey (CA), 378-385.

Balabanovic, M. and Shoham, Y. (1997). Fab: Content-based, collaborative recommendation. *Communications of the ACM* 40(3): 66-72.

Baltimore (2001). Baltimore Technologies. http://www.baltimore.com

Barnea, G. (1999). Intelligent Agent Communities. Manna.
19. January 2000. http://www.mannainc.com/downloads/intlagnt.zip

Bernstein, P., Hadzilacos, V. and Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems*. Reading (MA): Addison-Wesley.

Bertram, F. (2000). *VerBose – Verteilte Architektur für Benutzermodellierungssysteme*. Master Thesis, Dept. of Computer Science, University of Koblenz-Landau (Germany).

Bell, D. and Grimson, J. (1992). *Distributed Database Systems*. Reading (MA): Addison-Wesley.

Bigfoot (2000). Bigfoot. Bigfoot Communications. http://www.bigfoot.com

Billsus, D. and Pazzani, M. (2000). User Modeling for Adaptive News Access. *User Modeling and User-Adapted Interaction* 10(2/3): 147-180.

Blaze (2000). Blaze Software. 19. January 2000. http://www.blazesoftware.com

Bleymüller, J., Gehlert, G. and Gülicher, H. (1983). *Statistik für Wirtschaftswissenschaftler*. München: Vahlen.

Blue Martini (2000). Blue Martini Software. http://www.bluemartini.com

Boldon James (2000). LDAP-X toolkit. Protek Boldon James.
19. January 2000. http://www.bj.co.uk/ldapx.htm
30. September 2003. http://www.innosoft.com/ldap_survey/vendor/bj/sr_emdua.txt

Boeyen, S., Howes, T. and Richard, P. (1999a). Internet X.509 Public Key Infrastructure Operational Protocols – LDAP v2. http://www.ietf.org/rfc/rfc2559.txt?number=2559

Boeyen, S., Howes, T. and Richard, P. (1999b). Internet X.509 Public Key Infrastructure LDAP v2 Schema. http://www.ietf.org/rfc/rfc2587.txt?number=2587

Bowne (2000). Bowne & Co. http://www.bowne.com

Brajnik, G. and Tasso, C. (1994). A Shell for Developing Non-Monotonic User Modeling Systems. *International Journal of Human-Computer Studies* 40:31-62.

Breese, J., Heckerman, D. and Kadie, C. (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In Proceedings of the *Fourteenth Annual Conference on Uncertainty in Artificial Intelligence* (UAI-98). San Francisco (CA): Morgan Kaufmann, 43-52.

Breslau, L., Cao, P., Fan, L., Phillips, G. and Shenker, S. (1999). Web Caching and Zipf-Like Distributions: Evidence and Implications. In Proceedings of *INFOCOM'99*. IEEE Computer Society, 126-134. http://www.research.att.com/~breslau/pubs/zipf.ps.gz

BroadVision (2000). BroadVision. http://www.broadvision.com

Brusilovsky, P. (1996). Methods and Techniques of Adaptive Hypermedia. *User Modeling and User-Adapted Interaction* 6(2-3): 87-129.

Brusilovsky, P., Ritter, S. and Schwarz, E. (1997). Distributed Intelligent Tutoring on the Web. In B. du Boulay and R. Mizoguchi, eds., Proceedings of *AI-ED'97*. Amsterdam (Netherlands), 482-489.

Brusilovsky, P., Kobsa, A. and Vassileva, J., eds. (1998). *Adaptive Hypertext and Hypermedia*. Dordrecht: Kluwer Academic Publishers.

BusinessWeek (1999). Global 1000. BusinessWeek online. 3. March 2000. http://www.businessweek.com/1999/99_28/g1000.htm

Cadez, I., Heckerman, D., Meek, C., Smyth, P. and White, S. (2000). Visualization of navigation patterns on a Web site using model-based clustering. In Proceedings of the *Sixth ACM Conference on Knowledge Discovery and Data Mining*. New York (NY): ACM, 280-284.

Caglayan, A. and Snorrason, M. (1993). On the Relationship between Generalized Equality Clustering and ART 2 Neural Networks. *World Congress on Neural Networks*. Portland (OR).

Caglayan, A., Snorrason, M., Jacoby, J., Mazzu, J., Jones, R. and Kumar, K. (1997). Learn Sesame – a Learning Agent Engine. *Applied Artificial Intelligence* 11: 393-412.

Card, S., Mackinlay, J. and Shneiderman, B. (1999). Information Visualization. In S. Card, J. Mackinlay and B. Shneiderman, eds., *Readings in Information Visualization: Using Vision to Think*. San Francisco (CA): Morgan Kaufmann, 1-34.

Carroll, J., ed. (1995). *Scenario-based Design: Envisioning Work and Technology in System Development*. New York (NY): Wiley and Sons.

Carroll, J., ed. (2000). *Making Use: Scenario-based Design of Human-Computer Interactions*. Cambridge (MA): MIT Press.

Castano, S., Fugini, M., Martella, G. and Samarati, P. (1995). *Database Security*. Reading (MA): Addison-Wesley.

Chadwick, D. (1996). *Understanding X.500: The Directory*. London: Thomson.

Chandrasekhara, V. (1999). *JAMFrame – Java Agent Modules*. European Media Laboratory, Heidelberg (Germany).

Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D. and Sartin, M. (1999). Combining Content-Based and Collaborative Filters in an Online Newspaper. In Proceedings of the *ACM SIGIR '99 Workshop on Recommender Systems: Algorithms and Evaluation*. New York (NY): ACM. http://www.csee.umbc.edu/~ian/sigir99-rec/papers/claypool_m.ps.gz

CNN (2001). CNN.com. Cable News Network LP. http://www.cnn.com

Computer Scope (1999). 35 Percent of Surfing Time is Spent on 50 Sites. Computer Scope. http://www.nua.com/surveys/index.cgi?f=VS&art_id=905355323&rel=true

Condliff, M., Lewis, D., Madigan, D. and Posse, C. (1999). Bayesian Mixed-Effects Models for Recommender Systems. In Proceedings of the *ACM SIGIR '99 Workshop on Recommender Systems: Algorithms and Evaluation*. New York (NY): ACM. http://www.csee.umbc.edu/~ian/sigir99-rec/papers/condliff_m.ps.gz

Conklin, J. (1987). Hypertext: An Introduction and Survey. *IEEE Computer* September 1987: 17-41.

Cooperstein, D., Delhagen, K., Aber, A. and Levin, K. (1999). *Making Net Shoppers Loyal*. Cambridge (MA): Forrester Research.

Cost, S. and Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning* 10: 57-78.

Critical Path (2000). Critical Path. http://www.cp.net

Datta, A., Dutta, K., VanderMeer, D., Ramamritham, K. and Navathe, S. (2001). An Architecture to Support Scalable Online Personalization on the Web. *The VLDB Journal* 10: 104-117.

Date, C. (1986). *An Introduction to Database Systems*. Reading (MA): Addison-Wesley.

Dayal, U., Buchmann, A. and Mc Carthy, D. (1988). Rules are Objects too: A Knowledge Model for an Active Object-Oriented Database System. In K. Dittrich, ed., *Advances in Object-Oriented Database Systems,* Proceedings of the *2nd Int. Workshop on Object-Oriented Database Systems*. Berlin, Heidelberg: Springer-Verlag.

Deep Map (2001). Deep Map: Intelligent, Mobile, Multi-Media and Full of Knowledge. European Media Laboratory (EML). http://www.eml-development.de/english/Research/Memory/1

Deerwester, S., Dumais, S., Furnas, G., Landauer, T. and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science* 41(6): 391-407.

Dierks, T. and Allen, C. (1999). The TLS Protocol. http://www.ietf.org/rfc/rfc2246.txt?number=2246

Diffie, W. and Landau, S. (1998). *Privacy on the Line: The Politics of Wiretapping and Encryption*. Cambridge (MA): MIT Press.

DMA (2000). Direct Marketing Association. http://www.the-dma.org

Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis*. New York (NY): Wiley and Sons.

Duska, B., Marwood, D. and Feeley, M. (1997). The Measured Access of World Wide Web Proxy Caches. In Proceedings of the *1st USENIX Symposium on Internet Technologies and Systems*. Monterey (CA).

Eklund, J. and Brusilovsky, P. (1998). The Value of Adaptivity in Hypermedia Learning Environments: A Short Review of Empirical Evidence. In P. Brusilovsky and P. De Bra, eds., Proceedings of *Second Adaptive Hypertext and Hypermedia Workshop at the Ninth ACM International Hypertext Conference Hypertext'98*. Pittsburgh (PA), 11-17. http://wwwis.win.tue.nl/ah98/Eklund.html

Excite (2002). Excite Network Online Media Kit. Excite. http://www.excitenetwork.com/advertising/index/id/Directmarket|ListRental|3|1.html

EML (1999). Annual Report 1998/1999. Heidelberg (Germany): European Media Laboratory (EML).

EML (2000). Annual Report 2000. Heidelberg (Germany): European Media Laboratory (EML).

EPIC (2000). Electronic Privacy Information Center. http://www.epic.org

Finin, T. W. (1989). GUMS: A General User Modeling Shell. In A. Kobsa and W. Wahlster, eds., *User Models in Dialog Syste*ms. Berlin, Heidelberg: Springer-Verlag, 411-430.

Finin, T., Weber, J., Widerhold, G., Genesereth, M., Fritzson, R., McKay, D., McGuire, J., Pelavin, R., Shapiro, S. and Beck, C. (1993). Specification of the KQML Agent-Communication Language. http://www.cs.umbc.edu/kqml/papers/kqmlspec.ps

Fink, J. (1996). A Flexible and Open Architecture for the User Modeling Shell System BGP-MS. In S. Carberry, D. Chin and I. Zukerman, eds., Proceedings of *UM96: Fifth International Conference on User Modeling*. West Newton (MA): User Modeling Inc., 237-239.

Fink, J., Kobsa, A. and Nill, A. (1998). Adaptable and Adaptive Information Provision for All Users, Including Disabled and Elderly People. *The New Review of Hypermedia and Multimedia* 4: 163-188.

Fink, J. (1999). Transactional Consistency in User Modeling Systems. In J. Kay, ed.: *UM99 User Modeling: Proceedings of the Seventh International Conference*. Wien, New York (NY): Springer, 191-200.

Fink, J. and Kobsa, A. (2000). A Review and Analysis of Commercial User Modeling Servers for Personalization on the World Wide Web. *User Modeling and User-Adapted Interaction* 10(2-3): 209-249.

Fink, J. and Kobsa, A. (2002). User Modeling for Personalized City Tours. *Artificial Intelligence Review* 18: 33-74.

Fink, J., Noller, S., Koenemann, J. and Schwab, I. (2002). Putting Personalization into Practice. *Communications of the ACM* 45(5): 41-42.

FIPA (1998a). *FIPA 98 Specification Part 1: Agent Management*. Foundation for Intelligent Physical Agents (FIPA), Geneva (Switzerland). http://www.fipa.org/specifications/index.html

FIPA (1998b). *FIPA 98 Specification Part 8: Human Agent Interaction*. Foundation for Intelligent Physical Agents (FIPA), Geneva (Switzerland). http://www.fipa.org/specifications/index.html

Fisher, D. (1987). Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning* 2(2): 139-172.

Fisher, D. (1996). Iterative Optimization and Simplification of Hierarchical Clusterings. *Journal of Artificial Intelligence Research* 4: 147-179.

Forrester (2000). Forrester Research. http://www.forrester.com

FTC (2000). Federal Trade Commission. http://www.ftc.gov

Fuller, R. and de Graaf, J. (1996). Measuring User Motivation from Server Log Files. In Proceedings of the Microsoft Conference *Designing for the Web – Empirical Studies*. Redmond (WA). http://www.microsoft.com/usability/webconf/fuller/fuller.htm

Gawor, J. (1999). *LDAP Browser/Editor*. http://www-unix.mcs.anl.gov/~gawor/ldap

GMD AiS (2000). AI Publications & Links. GMD – German National Research Center for Information Technology, Institute for Autonomous intelligent Systems. http://ais.gmd.de/services/AI-Conferences/ai-journals.html

Goldberg, D., Nichols, D., Oki, B. and Terry, D. (1992). Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM* 35(12): 61-70.

Good, N., Schafer, J., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J. and Riedl, J. (1999). Combining Collaborative Filtering with Personal Agents for Better Recommendations. In Proceedings of the *1999 Conference of the American Association of Artificial Intelligence (AAAI-99)*. Cambridge (MA): MIT Press, 439-446. http://www.cs.umn.edu/Research/GroupLens/papers/pdf/aaai-99.pdf

Good, G. (2000). The LDAP Data Interchange Format (LDIF) – Technical Specification. http://www.ietf.org/rfc/rfc2849.txt?number=2849

Goscinski, A. (1991). *Distributed Operating Systems: The Logical Design*. Sydney: Addison-Wesley.

Grant, G. (1997). *Understanding Digital Signatures: Establishing Trust over the Internet and Other Networks*. New York (NY): McGraw-Hill.

Gray, J. (1981). The Transaction Concept: Virtues and Limitations. In Proceedings of the *Seventh Conference on Very Large Data Bases (VLDB'81)*. 144-154.

Gray, J. and Reuter, A. (1993). *Transaction Processing: Concepts and Techniques*. San Mateo (CA): Morgan Kaufmann.

Gribble, S. and Brewer, E. (1997). System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. In Proceedings of the *1st USENIX Symposium on Internet Technologies and Systems*. Monterey (CA).

Gustos (2000). Gustos Software. http://www.gustos.com

Härder, T. and Reuter, A. (1983). Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys* 15(4): 287-317.

Hagen, P., Manning, H. and Souza, R. (1999). *Smart Personalization*. Cambridge (MA): Forrester Research.

Herlocker, J., Konstan, J., Borchers, A. and Riedl, J. (1999). An Algorithmic Framework for Performing Collaborative Filtering. In M. Hearst, F. Gey and R. Tong, eds., Proceedings of the *22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York (NY): ACM, 230–237. http://www.cs.umn.edu/Research/GroupLens/papers/pdf/algs.pdf

Heuer, A. and Scholl, M. (1991). Principles of Object-Oriented Query Languages. In H.-J. Appelrath, ed., Proceedings *GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft" (BTW'91)*. Berlin, Heidelberg: Springer-Verlag, 178-197.

Heuer, A. and Saake, G. (1995). *Datenbanken – Konzepte und Sprachen*. Bonn: Thomson.

Hill, W., Stead, L., Rosenstein, M. and Furnas, G. (1995). Recommending and evaluating choices in a virtual community of use. In Proceedings of *ACM CHI'95 Conference on Human Factors in Computing Systems*. New York (NY): ACM, 194-201.

HNC (2000). HNC Software. http://www.hnc.com

Hof, R., Green, H. and Himmelstein, L. (1998). Now it's YOUR WEB. *Business Week*, October 5: 68-74.

Howes, T. and Smith, M. (1995). The LDAP Application Program Interface. http://www.ietf.org/rfc/rfc1823.txt?number=1823

Howes, T., Kille, S., Yeong, W. and Robbins, C. (1995). The String Representation of Standard Attribute Syntaxes. http://www.ietf.org/rfc/rfc1778.txt?number=1778

Howes, T. (1997). The String Representation of LDAP Search Filters. http://www.ietf.org/rfc/rfc2254.txt?number=2254

Howes, T. and Smith, M. (1997a). The LDAP URL Format. http://www.ietf.org/rfc/rfc2255.txt?number=2255

Howes, T. and Smith, M. (1997b). *Ldap: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*. Indianapolis (IN): Macmillan.

Howes, T., Smith, M. and Good, G. (1999). *Understanding and deploying LDAP directory services*. Indianapolis (IN): Macmillan.

humanIT (2001). humanIT. http://www.humanit.de

IBM (2000a). Portals – Commerce - Personalization. IBM. http://www-3.ibm.com/software/info/portal-commerce/index.jsp

IBM (2000b). Lotus Notes. IBM. http://www.lotus.com/products/r5web.nsf/webpi/Notes?opendocument

IBM (2000c). DSSeries. Directory Server. IBM. http://www-3.ibm.com/software/network/help-directory/

ICONOCAST (1999). Brand conversion. ICONOCAST.
29. January 2000. http://www.iconocast.com/issue/1999102102.html
30. September 2003. http://groups.yahoo.com/group/iconocast/message/38

IETF (2000). IETF RFC Page. IETF (Internet Engineering Task Force). http://www.ietf.org/rfc.html

Informix (2000). Informix.
29. July 2000. http://www.informix.com
30. September 2003. http://www-3.ibm.com/software/data/informix/

Innosoft (2000). Innosoft. http://www.innosoft.com

Inprise (2000). Inprise.
20. February 2000. http://www.inprise.com
30. September 2003. http://www.borland.com/

iPlanet (2000a). iPlanet E-Commerce Solutions. iPlanet.
15. March 2000. http://www.iplanet.com/products/index.html
30. September 2003. http://wwws.sun.com/software/

iPlanet (2000b). iPlanet Directory Server. iPlanet.
15. March 2000. http://www.iplanet.com/products/infrastructure/dir_security/dir_srvr
30. September 2003.
http://wwws.sun.com/software/product_categories/directory_servers_identity_mgmt.html

ISO (1989). Database Languages -- SQL. ISO/IEC 9075:1989. International Standardization Organization.

ISO (1998a). Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) -- Part12: Presentation of Information. ISO 9241-12:1998.

ISO (1998b). Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) -- Part13: User guidance. ISO 9241-13:1998.

ISO (1999). Database Languages -- SQL. ISO/IEC 9075:1999. International Standardization Organization.

ITU-T (1997). Information Technology – Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. X.680. International Telecommunication Union.

ITU-T (2001a). Information Technology – Open Systems Interconnection – The Directory: Overview of Concepts, Models and Services. X.500. International Telecommunication Union.

ITU-T (2001b). Information Technology – Open Systems Interconnection – The Directory: Models. X.501. International Telecommunication Union.

ITU-T (2001c). Information Technology – Open Systems Interconnection – The Directory: Selected Attribute Types. X.520. International Telecommunication Union.

ITU-T (2001d). Information Technology – Open Systems Interconnection – The Directory: Selected Object Classes. X.521. International Telecommunication Union.

IVW (2001). Online Usage Data November 2001 (in German). IVW. http://www.ivwonline.de/ausweisung/suchen.php

Jameson, A. (1999). User-Adaptive Systems: An Integrative Overview. Tutorial presented at the *Seventh International Conference on User Modeling* and at the *Sixteenth International Joint Conference on Artificial Intelligence*. http://dfki.de/~jameson/um99-tutorial-index.html

Jennings, A. and Higuchi, H. (1993). A user model neural network for a personal news service. *User Modeling and User-Adapted Interaction* 3(1): 1-25.

Jörding (1999). Temporary User Modeling for Adaptive Product Presentations in the Web. In J. Kay, ed.: *UM99 User Modeling: Proceedings of the Seventh International Conference*. Wien, New York (NY): Springer, 333-334.

Jupiter (2000). Jupiter. http://www.jup.com

Jupitermedia (2002). MSN Hits 300 Million Unique Monthly Users. Jupitermedia. http://cyberatlas.internet.com/big_picture/traffic_patterns/article/0,,5931_1457661,00.html

Kaul, E. (1999). *Soziokulturelle Kategorisierung der Touristen in Heidelberg*. Master Thesis, Geographical Institute University of Heidelberg (Germany).

Kay, J. (1995). The um Toolkit for Reusable, Long Term User Models. *User Modeling and User-Adapted Interaction* 4(3): 149-196.

Keung, S. and Abbott, S. (1998). LDAP Server Performance Report. http://www.bnelson.com/sizing

KDnuggets (2000). KDnuggets: Data Mining, Web Mining, and Knowledge Discovery Guide. KDnuggets. http://www.kdnuggets.com

Kille, S. (1995). A String Representation of Distinguished Names. http://www.ietf.org/rfc/rfc1779.txt?number=1779

Kobsa, A. and Wahlster, W., eds. (1989). *User Models in Dialog Systems.* Berlin, Heidelberg: Springer-Verlag.

Kobsa, A., Müller, D. and Nill, A. (1994). KN-AHS: An Adaptive Hypertext Client of the User Modeling System BGP-MS. In Proceedings of the *Fourth International Conference on User Modeling*. Hyannis (MA), 99-105. Reprinted in M. Maybury and W. Wahlster, eds. (1998). *Intelligent User Interfaces*. San Mateo (CA): Morgan Kaufman, 372-378. http://www.ics.uci.edu/~kobsa/papers/1994-UM94-kobsa.pdf

Kobsa, A. and Pohl, W. (1995). The User Modeling Shell System BGP-MS. *User Modeling and User-Adapted Interaction* 4(2): 59-106.

Kobsa, A., Pohl, W. and Fink, J. (1996). A Standard for the Performatives in the Communication between Applications and User Modeling Systems (Draft). http://www.ics.uci.edu/~kobsa/papers/1996-kobsa-pohl-fink-rfc.pdf

Kobsa, A. (2000). User Modeling, Privacy, and Security. Invited talk held at the *First International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, Trento (Italy). http://www.ics.uci.edu/~kobsa/talks/privsecum/index.htm

Kobsa, A. (2001a). Generic User Modeling Systems. *User Modeling and User-Adapted Interaction* 10(4), Ten Year Anniversary Issue, 49-63.

Kobsa, A. (2001b). Tailoring Privacy to Users' Needs. In M. Bauer, P. Gmytrasiewicz and J. Vassileva, eds., Proceedings of the *Eighth International Conference on User Modeling*. Berlin, Heidelberg: Springer-Verlag, 303-313. http://www.ics.uci.edu/~kobsa/papers/2001-UM01-kobsa.pdf

Kobsa, A. (2001c). ICS 280: Advanced Topics in Information Visualization. http://www.ics.uci.edu/~kobsa/courses/ICS280/01S.htm

Kobsa, A., Koenemann, J. and Pohl, W. (2001). Personalized Hypermedia Presentation Techniques for Improving Customer Relationships. *The Knowledge Engineering Review* 16(2): 111-155. http://www.ics.uci.edu/~kobsa/papers/2001-KER-kobsa.pdf

Kobsa, A. and Fink, J. (2003). Performance Evaluation of User Modeling Servers under Real-World Workload Conditions. In P. Brusilovsky, A. Corbett and F. de Rosis, eds., Proceedings of the *Ninth International Conference on User Modeling*. Berlin, Heidelberg: Springer-Verlag, 143-153.

Kobsa, A. and Schreck, J. (2003). Privacy through Pseudonymity in User-Adaptive Systems. *ACM Transactions on Internet Technology* 3(2): 149-183. http://www.ics.uci.edu/~kobsa/papers/2003-TOIT-kobsa.pdf

Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L. and Riedl, J. (1997). GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM* 40(3): 77-87. http://www.acm.org/pubs/citations/journals/cacm/1997-40-3/p77-konstan

Kristol, D. and Montulli, L. (1997). HTTP State Management Mechanism. http://www.ietf.org/rfc/rfc2109.txt?number=2109

Kummerfeld, R. and Kay, J. (1997). Remote Access Protocols for User Modelling. In *Proceedings and Resource kit for Workshop User Models in the Real World*. Chia Laguna (Sardinia), 12-15.

Labrou, Y. and Finin, T. (1997). *Proposal for a new KQML Specification*, TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County. http://www.csee.umbc.edu/~jklabrou/publications/tr9703.pdf

Lieberman, H. (1995). Letizia: An Agent That Assists Web Browsing. In Proceedings of the *International Joint Conference on Artificial Intelligence*, Montreal (Canada). San Mateo (CA): Morgan Kaufmann, 924-929. http://lcs.www.media.mit.edu/people/lieber/Lieberary/Letizia/Letizia-Intro.html

Lilien, G., Kotler, P. and Moorthy, K. (1992). *Marketing Models*. Englewood Cliffs (NJ): Prentice Hall.

Loshin, P. (2000). *Big Book of Lightweight Directory Access Protocol (LDAP) RFCs*. San Diego (CA): Morgan Kaufmann.

Lucent (2000). Lucent Technologies. http://www.lucent.com

Machado, I., Martins, A. and Paiva, A. (1999). One for All and All in One: A Learner Modeling Server in a Multi-Agent Platform. In J. Kay, ed., *UM99 User Modeling: Proceedings of the Seventh International Conference*. Wien, New York (NY): Springer, 211-221.

Macromedia (2000). LikeMinds. Macromedia.
5. December 2000. http://www.macromedia.com/products/likeminds/

Magedanz, T. (1997). Mobile Agents – An Overview. Tutorial presented at the *ACTS IS&N Conference*, Cernobbio (Como), Italy.

Malaka, R. (1999). Deep Map: The Multilingual Tourist Guide. *C-Star Workshop*, Schwetzingen (Germany). http://www.eurescom.de/~pub/fusenetd/Malaka.pdf

Malaka, R. and Zipf, A. (2000). DEEP MAP – Challenging IT Research in the Framework of a Tourist Information System. In D. Fesenmaier, S. Klein and D. Buhalis, eds., *Information and Communication Technologies in Tourism 2000: Proceedings of ENTER 2000*. Wien, New York (NY): Springer, 15-27.

Manna (1999a). FrontMind for Marketing. Manna.
19. January 2000. http://www.mannainc.com/downloads/fmwhite.zip

Manna (1999b). Automated Distributed Intelligence. Manna.
19. January 2000. http://www.mannainc.com/downloads/adiwhite.zip

Manna (2000a). Manna. 19. January 2000. http://www.mannainc.com

Manna (2000b). FrontMind. Manna.
4. April 2000. http://www.mannainc.com/downloads/whitepaper.zip

Manna (2000c). Online Personalization for E-commerce: The Manna Advantage. Manna.
4. April 2000. http://www.mannainc.com/downloads/personalization.zip

Manna (2000d). FrontMind Components. Manna.
4. April 2000. http://www.mannainc.com/products_stage_technical.html

Marketing 1to1 (2000). Marketing 1to1 – Marketplace1to1. Peppers and Rogers Group.
5. March 2000. http://search.marketplace1to1.com

Martin, J. (1983). *The Data-Base Environment*. Englewood Cliffs (NJ): Prentice Hall.

Mayfield, J., Labrou, Y. and Finin, T. (1996). Evaluation of KQML as an Agent Communication Language. In M. Wooldridge, J. Müller and M. Tambe, eds., *Intelligent Agents Volume II -- Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages*. Berlin, Heidelberg: Springer-Verlag, 347-360.

McTear, M. (1993). User Modelling for Adaptive Computer Systems: a Survey. *Artificial Intelligence Review* 7(3-4): 157-184.

Microsoft (2000a). Internet Explorer. Microsoft. http://www.microsoft.com/windows/ie

Microsoft (2000b). Active Directory Service Interfaces. Microsoft. http://www.microsoft.com/windows2000/server/evaluation/news/bulletins/adextension.asp

Microsoft (2000c). Active Directory Architecture. Microsoft. http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/ad/windows2000/deploy/projplan/adarch.asp

Microsoft (2000d). Product and Technology Catalog. Microsoft. http://www.microsoft.com/products

Millhouse, C., Brash, C. and Chapple, D. (2000). *E-CRM: Personalisation Technologies for the Web*. London: Ovum.

Mindcraft (2001). Mindcraft. http://www.mindcraft.com

Mitchell, T. (1997). *Machine learning*. New York (NY): McGraw-Hill.

Mohr, S. (1999). *Designing Distributed Applications*. Birmingham: Wrox Press.

Mooney, R. and Roy, L. (1999). Content-Based Book Recommending Using Learning for Text Categorization. In Proceedings of the *Workshop on Recommender Systems: Algorithms and Evaluation at the 22nd International Conference on Research and Development in Information Retrieval*. Berkeley (CA). http://www.csee.umbc.edu/~ian/sigir99-rec/papers/mooney_r.ps.gz

Moore, J. and Paris, C. (1992). Exploiting User Feedback to Compensate for the Unreliability of User Models. *User Modeling and User-Adapted Interaction* 2(4): 331-365.

MSNBC (2001). MSNBC. http://www.msnbc.com

Myers, J. (1997). Simple Authentication and Security Layer (SASL). http://www.ietf.org/rfc/rfc2222.txt?number=2222

NAI (2001). *Self-Regulatory Principles for Online Preference Marketing by Network Advisers*. Network Advertising Initiative. http://www.networkadvertising.org/images/NAI_Principles.pdf

Nelson, B. (2002). Sizing Guide for Netscape Directory Server. http://www.bnelson.com/sizing/doc2/Directory4_0-SizingGuide.html

Net Perceptions (2000). Net Perceptions. http://www.netperceptions.com

Netegrity (2001). Netegrity. http://www.netegrity.com

NetRatings (2000). NetRatings. http://www.netratings.com

NetRatings (2002). Top 10 Web Properties for the Month of October 2002. NetRatings. 15. November 2002. http://epm.netratings.com/de/web/NRpublicreports.toppropertiesmonthly.html

Netscape (2000a). Netscape Browsers. Netscape. http://www.netscape.com/browsers

Netscape (2000b). Netscape Directory SDK. Netscape. http://developer.netscape.com/tech/directory/downloads.html

Nielsen, J. (1990). The Art of Navigating through Hypertext. *Communications of the ACM* 33(3): 296-310.

Nielsen, J. (1993). *Usability Engineering*. San Diego (CA): Academic Press.

Nielsen, J. (1996). Top Ten Mistakes in Web Design. http://www.useit.com/alertbox/9605.html

Nielsen, J. (1997). Zipf Curves and Website Popularity. http://www.useit.com/alertbox/zipf.html

Nielsen Media (2000). Nielsen Media Research. http://www.nielsenmedia.com

Novell (2000). Novell eDirectory. Novell. http://www.novell.com/products/nds/

O'Conner, M. and Herlocker, J. (1999). Clustering Items for Collaborative Filtering. In Proceedings of the *Workshop on Recommender Systems: Algorithms and Evaluation at the 22nd International Conference on Research and Development in Information Retrieval*. Berkeley (CA). http://www.csee.umbc.edu/~ian/sigir99-rec/papers/oconner_m.pdf

Oard, D. (1997). The State of the Art in Text Filtering. *User Modeling and User-Adapted Interaction* 7(3): 141-178.

ObjectSpace (2000). ObjectSpace. 15. August 2002. http://www.objectspace.com
30. September 2003. http://www.recursionsw.com/

Oblix (2000). Oblix. http://www.oblix.com

OMG (2001). Object Management Group (OMG). http://www.omg.org

OpenLDAP (2000). OpenLDAP. http://www.openldap.org

Open Sesame (1998). Open Sesame. 16. July 1998. http://www.opensesame.com

Open Sesame (1999). Open Sesame. 26. November 1999. http://www.opensesame.com

Open Sesame (2000). Open Sesame. 18. June 2000. Bowne & Co. http://www.opensesame.com

Oppermann, R. (1994). *Adaptive User Support - Ergonomic Design of Manually and Automatically Adaptable Software*. Hillsdale (NJ): Lawrence Erlbaum.

Orfali, R., Harkey, D. and Edwards, J. (1994). *Essential Client/Server Survival Guide*. New York (NY), Singapore: Wiley and Sons.

Orwant, J. (1995). Heterogeneous Learning in the Doppelgänger User Modeling System. *User Modeling and User-Adapted Interaction* 4(2): 107-130.

Padmanabhan, V. and Qiu, L. (2000). The Content and Access Dynamics of a Busy Web Site: Findings and Implications. In Proceedings of *ACM SIGCOMM 2000*. New York (NY): ACM, 111-123.

Paiva, A. and Self, J. (1995). TAGUS – A User and Learner Modeling Workbench. *User Modeling and User-Adapted Interaction* 4(3): 197-226.

Paliouras, G., Karkaletsis, V., Papatheodorou, C. and Spyropoulos, C. (1999). Exploiting Learning Techniques for the Acquisition of User Stereotypes and Communities. In J. Kay, ed., *UM99 User Modeling: Proceedings of the Seventh International Conference*. Wien, New York (NY): Springer, 169-178.

Patrick, A. and Black, A. (1996). Implications of Access Methods and Frequency of Use for the National Capital Freenet. 18. October 2000.
http://debra.dgbt.doc.ca/services-research/survey/connections/

Pazzani, M. and Billsus, D. (1997). Learning and revising user profiles: The identification of interesting Web sites. *Machine Learning* 27: 313-331.

Peppers, D. and Rogers, M. (1993). *The One to One Future: Building Relationships One Customer at a Time.* New York (NY): Currency Doubleday.

Peppers, D. and Rogers, M. (1997). *Enterprise One to One: Tools for Competing in the Interactive Age.* New York (NY): Currency Doubleday.

Peppers, D., Rogers, M. and Dorf, B. (1999). *The One to One Fieldbook.* New York (NY): Currency Doubleday.

Persistent (2000). Persistent Systems Private Limited. http://www.pspl.co.in

Personalization (2000). personalization.com. 18. June 2000. http://www.personalization.com

Pohl, W. and Höhle, J. (1997). Mechanisms for Flexible Representation and Use of Knowledge in User Modeling Shell Systems. In A. Jameson, C. Paris and C. Tasso, eds., *User Modeling: Proceedings of the Sixth International Conference*. Wien, New York (NY): Springer, 403-414.

Pohl, W. (1998). Logic-Based Representation and Reasoning for User Modeling Shell Systems. Sankt Augustin (Germany): infix.

Pohl, W. and Nick, A. (1999). Machine Learning and Knowledge-Based User Modeling in the LaboUr Approach. In J. Kay, ed., *UM99 User Modeling: Proceedings of the Seventh International Conference*. Wien, New York (NY): Springer, 179-188.

Pohl, W., Schwab, I. and Koychev, I. (1999). Learning About the User: A General Approach and Its Application. In Proceedings of *IJCAI'99 Workshop Learning About Users*. Stockholm (Sweden). http://www.cs.rutgers.edu/ml4um/mirrors/lau-1999/papers/pohl.ps

Pope, A. (1997). *The Corba Reference Guide: Understanding the Common Object Request Broker Architecture*. Sydney: Addison-Wesley.

Popp, H. and Lödel, D. (1996). Fuzzy Techniques and User Modeling in Sales Assistants. *User Modeling and User-Adapted Interaction* 5(3-4): 349-370.

Qualcomm (2000). Qualcomm. http://www.qualcomm.com

Quinlan, J. (1986). Induction of decision trees. *Machine Learning* 1(1): 81-106.

Reagle, J. and Cranor, L. (1999). The Platform for Privacy Preferences. *Communications of the ACM* 42(2): 48-55.

Reichheld, F. (1996). *The Loyalty Effect*. Boston (MA): Harvard Business School Press.

Resnick, P., Iacovou, N., Sushak, M., Bergstrom, P. and Riedl, J. (1994). GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In Proceedings of the *Conference on Computer Supported Cooperative Work*. New York (NY): ACM, 175-186. http://www.acm.org/pubs/citations/proceedings/cscw/192844/p175-resnick/

Rich, E. (1979). User Modeling via Stereotypes. *Cognitive Science* 3: 329-354.

Rich, E. (1983). Users are Individuals: Individualizing User Models. *Journal of Man-Machine Studies* 18: 199-214.

Rich, E. (1989): Stereotypes and User Modeling. In A. Kobsa and W. Wahlster, eds.: User Models in Dialog Systems. Berlin, Heidelberg: Springer-Verlag, 35-51.

RightPoint (2000). RightPoint Software. 8. February 2000. http://www.rightpoint.com

Rozanski, H., Bollman, G. and Lipman, M. (2000). Seize the Occasion – Usage-based Segmentation for Internet Marketers. McLean (VA): Booz-Allen & Hamilton. 26. March 2000. http://www.strategy-business.com/enews/032001/03-20-01_eInsight.pdf

Russell, S. (2000). AI on the Web. http://www.cs.berkeley.edu/~russell/ai.html

Ryan, V., Seligman S. and Lee R. (1999a). Schema for Representing Java(tm) Objects in an LDAP Directory. http://www.ietf.org/rfc/rfc2713.txt?number=2713

Ryan, V., Seligman S. and Lee R. (1999b). Schema for Representing CORBA Object References in an LDAP Directory. http://www.ietf.org/rfc/rfc2714.txt?number=2714

Saake, G., Türker, C. and Schmitt, I. (1997). *Objektdatenbanken – Konzepte, Sprachen, Architekturen*. Bonn: Thomson.

Salton, G. and McGill, M. (1983). *Introduction to Modern Information Retrieval*. New York (NY): McGraw-Hill.

SAP (2001). SAP. http://www.sap.com

Sarwar, B., Karypis, G., Konstan, J. and Riedl, J. (2000). Analysis of Recommendation Algorithms for E-Commerce. In Proceedings of the *2nd ACM Conference on Electronic Commerce*. New York (NY): ACM, 158–167. http://www.cs.umn.edu/Research/GroupLens/papers/pdf/ec00.pdf

Schafer, J., Konstan, J. and Riedl, J. (1999). Recommender Systems in Electronic Commerce. In Proceedings of the *ACM Conference on Electronic Commerce (EC-99)*. New York (NY): ACM, 158-166. http://www.cs.umn.edu/Research/GroupLens/papers/pdf/ec-99.pdf

Schmidt, D. (1994). A Domain Analysis of Network Daemon Design Dimensions. *C++ Report* 6(3/4).

Schreck, J. (2003). *Security and Privacy in User Modeling*. Dordrecht: Kluwer Academic Publishers (forthcoming).

Schrock, R. (1999). AltaVista Announcement. Compaq. http://www.compaq.com/newsroom/presspaq/012699/schrock.html

Schwab, I. and Pohl, W. (1999). Learning Information Interests from Positive Examples. In *UM99 Workshop Machine Learning for User Modeling*. Banff (Canada). 16. October 1999. http://fit.gmd.de/~schwab/Papers/ml4um-um99.ps

Schwab, I., Kobsa, A. and Koychev, I. (2000). Learning about Users from Observation. In *Adaptive User Interfaces: Papers from the 2000 AAAI Spring Symposium*. Menlo Park (CA): AAAI Press, 102-106. http://www.ics.uci.edu/~kobsa/papers/2000-AAAI-kobsa.pdf

Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating "word of mouth". In Proceedings of *ACM CHI'95 Conference on Human Factors in Computing Systems*. New York (NY): ACM, 210-217.

Shneiderman, B. (1987). *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* New York (NY), Tokyo: Addison-Wesley.

Shukla, S. and Deshpande, A. (2000). LDAP Directory Services – Just Another Database Application? In C. Weidong, J. Naughton and P. Bernstein, eds., Proceedings of the *2000 ACM SIGMOD International Conference on Management of Data*. New York (NY): ACM, 580.

Singh, J. (2000). Privacy, Information, and Consumer Value. ART Technology Group. 23. June 2000. http://www.atg.com/news/white-papers/privacy.html

Smith, R. (1997). *Internet Cryptography*. Reading (MA): Addison-Wesley.

Smith, M. (2000). Definition of the inetOrgPerson LDAP Object Class. http://www.ietf.org/rfc/rfc2798.txt?number=2798

Snorrason, M. and Caglayan, A. (1994). Generalized ART2 Algorithms. *World Congress on Neural Networks*. San Diego (CA).

Soboroff, I., Nicholas, C. and Pazzani, M. (1999). Workshop on Recommender Systems: Algorithms and Evaluation – Workshop Summary. http://www.csee.umbc.edu/~ian/sigir99-rec/summary.html

Sparck Jones, K. (1972). A Statistical Interpretation of Term Specificity and its Application to Retrieval. *Journal of Documentation* 28: 11-21.

Specht, M. (1998). Empirical Evaluation of Adaptive Annotation in Hypermedia. In Proceedings of the *ED-MEDIA98*. Freiburg (Germany), 1327-1332.

Specht, M. and Kobsa, A. (1999). Interaction of Domain Expertise and Interface Design in Adaptive Educational Hypermedia. In Proceedings of the *Second Workshop on Adaptive Systems and User Modeling on the World Wide Web at WWW-8 and UM99.* Toronto (Canada) and Banff (Canada), 89-93. http://www.ics.uci.edu/~kobsa/papers/1999-WWW8UM99-kobsa.pdf

Spiliopoulou, M. and Faulstich, L. (1999). A Tool for Web Utilization Analysis. In Proceedings of the *International Workshop on the Web and Databases – WebDB'98.* Berlin, Heidelberg: Springer-Verlag, 184-203.

Stevens, W. (1990). *UNIX Network Programming*. Englewood Cliffs (NJ): Prentice Hall.

Stokes, E., Byrne, D., Blakley, B. and Behera, P. (2000). Access Control Requirements for LDAP. http://www.ietf.org/rfc/rfc2820.txt?number=2820

Sun (2000a). Java Technology. Sun. http://wwws.sun.com/software/java/index.html

Sun (2000b). Java Naming and Directory Interface (JNDI). Sun. http://java.sun.com/products/jndi/index.html

Sun (2000c). Sun Directory Services. Sun. 13. September 2000. http://www.sun.com/solstice/telecom/LDAP.html

Sun (2002a). Sun ONE Directory Server. Sun. http://wwws.sun.com/software/products/directory_srvr/home_directory.html

Sun (2002b). Sun Microsystems – Entry-Level Servers. Sun. http://www.sun.com/servers/entry/

Switchboard (2000). Switchboard: The Internet Directory – YellowPages – WhitePages – PhoneBook. Switchboard. http://www.switchboard.com

Tanenbaum, A. (1992). *Modern Operating Systems*. Englewood Cliffs (NJ): Prentice Hall.

Tanenbaum, A. (1995). *Distributed Operating Systems*. Englewood Cliffs (NJ): Prentice Hall.

Truog, D., Bernoff, J., Ritter, T. and Goldman, H. (1999). *Centralize Access Control Now.* Cambridge (MA): Forrester Research.

TRUSTe (2000). TRUSTe. http://www.truste.org

University of Michigan (2000a). Lightweight Directory Access Protocol – Clients. University of Michigan. 13. September 2000. http://www.umich.edu/~dirsvcs/ldap/ldclients.html

University of Michigan (2000b). Slapd Manual Page. University of Michigan. 13. September 2000. http://www.umich.edu/cgi-bin/ldapman?slapd

VanderMeer, D., Dutta, K. and Datta, A. (2000). Enabling Scalable Online Personalization on the Web. In Proceedings of the *Second ACM Conference on Electronic Commerce*. New York (NY): ACM, 185-196.

Vassileva, J., McCalla, G. and Greer, J. (2003). Multi-Agent Multi-User Modeling in I-Help. *User Modeling and User-Adapted Interaction* (forthcoming).

Vignette (2000). Vignette. http://www.vignette.com

W3C (2000). Platform for Privacy Preferences (P3P) Project. W3C. http://www.w3.org/P3P

Wahl, M. (1997). A Summary of the X.500 (96) User Schema for use with LDAP v3. http://www.ietf.org/rfc/rfc2256.txt?number=2256

Wahl, M., Howes, T. and Kille, S. (1997a). Lightweight Directory Access Protocol (v3). http://www.ietf.org/rfc/rfc2251.txt?number=2251

Wahl, M., Coulbeck, A., Howes, T. and Kille, S. (1997b). Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions. http://www.ietf.org/rfc/rfc2252.txt?number=2252

Wahl, M., Kille, S. and Howes, T. (1997c). Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names. http://www.ietf.org/rfc/rfc2253.txt?number=2253

Wang, X., Schulzrinne, H., Kandlur, D. and Verma, D. (2000). Measurement and Analysis of LDAP Performance. In Proceedings of the *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. New York (NY): ACM, 156-165. http://www.cs.columbia.edu/~xinwang/public/paper/ldap_sigmetrics.pdf

Weber, G. and Specht, M. (1997). User Modeling and Adaptive Navigation Support in WWW-Based Tutoring Systems. In A. Jameson, C. Paris and C. Tasso, eds., *User Modeling: Proceedings of the Sixth International Conference*. Wien, New York (NY): Springer, 289-300.

WebGuide (2001). EML Deep Map – GIS – WebGuide. European Media Laboratory (EML). 10. August 2001. http://www.eml.org/english/research/deepmap/deepgis/webguide.html

Weltman, R., Tomlinson, C., Kekic, M., Sonntag, S., Sermersheim, J., Smith, M. and Howes, T. (2001). The Java LDAP Application Program Interface. http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldap-java-api-18.txt

Wilcox, M. (1999). *Implementing LDAP*. Birmingham: Wrox Press.

Wolman, A., Voelker, G., Sharma, N., Cardwell, N., Brown, M., Landray, T., Pinnel, D., Karlin, A. and Levy, H. (1999a). Organization-based Analysis of Web-Object Sharing and Caching. In Proceedings of the *2nd USENIX Symposium on Internet Technologies and Systems*. Boulder (CO).

Wolman, A., Voelker, G., Sharma, N., Cardwell, N., Brown, M., Landray, T., Pinnel, D., Karlin, A. and Levy, H. (1999b). On the Scale and Performance of Cooperative Web Proxy Caching. In Proceedings of the *17th ACM Symposium on Operating System Principles*. New York (NY): ACM, 16-31.

Woods, E. and Kyral, E. (1997). *Ovum evaluates : Data Mining*. London: Ovum.

Yaacovi, Y., Wahl, M. and Genovese, T. (1999). Lightweight Directory Access Protocol (v3): Extensions for Dynamic Directory Services. http://www.ietf.org/rfc/rfc2589.txt?number=2589

Yahoo! (2001). Yahoo!. http://www.yahoo.com

Yeong, W., Howes, T. and Kille, S. (1995). Lightweight Directory Access Protocol. http://www.ietf.org/rfc/rfc1777.txt?number=1777

Yimam Seid, D. and Kobsa, A. (2003). Expert Finding Systems for Organizations: Problem and Domain Analysis and the DEMOIR Approach. *Journal of Organizational Computing and Electronic Commerce* 13(1): 1-24. http://www.ics.uci.edu/~kobsa/papers/2003-JOCEC-kobsa.pdf

Young, A. (1995). Connection-Less Lightweight X.500 Directory Access Protocol. http://www.ietf.org/rfc/rfc1798.txt?number=1798

Zehnder, C. (1985). *Informationssysteme und Datenbanken*. Stuttgart: Teubner.

Zipf, G. (1949). *Human Behavior and the Principle of Least Effort*. Reading (MA): Addison-Wesley.

Zukerman, I., Albrecht, D. and Nicholson, A. (1999). Predicting Users' Requests on the WWW. In J. Kay, ed., *UM99 User Modeling: Proceedings of the Seventh International Conference*. Wien, New York (NY): Springer, 275-284.