To appear in User Modeling and User-Adapted Interaction: The Journal of Personalization Research

An LDAP-Based User Modeling Server and its Evaluation[*]

Alfred Kobsa
Donald Bren School of Information and Computer Sciences
University of California
Irvine, CA, U.S.A.
kobsa@uci.edu

Josef Fink
Department of Computer and Engineering Sciences
University of Applied Sciences
Frankfurt, Germany
jfink@fb2.fh-frankfurt.de

Abstract

Representation components of user modeling servers have been traditionally based on simple file structures and database systems. We propose directory systems as an alternative, which offer numerous advantages over the more traditional approaches: international vendor-independent standardization, demonstrated performance and scalability, dynamic and transparent management of distributed information, built-in replication and synchronization, a rich number of pre-defined types of user information, and extensibility of the core representation language for new information types and for data types with associated semantics. Directories also allow for the virtual centralization of distributed user models and their selective centralized replication if better performance is needed.

We present UMS, a user modeling server that is based on the Lightweight Directory Access Protocol (LDAP). UMS allows for the representation of different models (such as user and usage profiles, and system and service models), and for the attachment of arbitrary components that perform user modeling tasks upon these models. External clients such as user-adaptive applications can submit and retrieve information about users. We describe a simulation experiment to test the runtime performance of this server, and present a theory of how the parameters of such an experiment can be derived from empirical web usage research. The results show that the performance of UMS meets the requirements of current small and medium websites already on very modest hardware platforms, and those of very large websites on an entry-level business server configuration.

Keywords: user modeling server, directory server, LDAP, architecture, evaluation, performance, scalability

[*] The UMUAI managing editor for this paper was Sandra Carberry, University of Delaware.

1 Introduction and Overview

For nearly twenty years, researchers have been developing generic user modeling systems (which have been called ‘user modeling shell systems’ and more recently ‘user modeling servers’ [Kobsa 2001]). Such systems facilitate the development of user-adaptive applications through ‘built-in’ core user modeling functionality that these applications can utilize for providing user-adaptive services. In addition to these research prototypes, a number of commercial systems have been recently put on the market (and a far larger number advertised on short-lived websites) that aim at providing such functionality for a specific kind of user-adaptive applications, namely web-based customer relationship management systems [Fink and Kobsa 2000; Kobsa et al. 2001].

Early user modeling shell systems were for the most part developed in the area of Artificial Intelligence. Quite in its tradition, user models were generally stored in simple flat files, or in so-called knowledge representation systems which themselves were implemented in (indexed) files or databases. More recent user modeling servers predominantly use database systems for storing information about users, and the recent commercial systems store user models in fairly sophisticated database management systems.

In this paper, we will present a user modeling server that is based on a very different type of data repository, namely directory systems. The use of directories for user modeling systems has already been considered, and dismissed, by [Kummerfeld and Kay 1997]. We will however argue that the reasons for their rejection do not hold true any more today. Rather, directory systems offer a number of significant advantages over database systems that should make them the data storage of choice for user modeling servers in the currently prevailing application scenarios for user modeling.

In Section 2 of this paper, we will give a brief review of knowledge and data base management systems that have traditionally served as data repositories of generic user modeling systems. Section 3 introduces the alternative concept of directory systems, and specifically the Lightweight Directory Access Protocol (LDAP). We discuss five dimensions along which directory systems excel over database systems for user modeling purposes, as well as their derivative ability to implement so-called virtually centralized distributed user models. We will also describe a sixth dimension along which databases fare better than directories, but this is (at least for now) not very relevant for user modeling. Section 4 presents the architecture of UMS, an LDAP-based user modeling server that was used successfully in two different application domains. The proposed architecture also includes solutions for scalability (to a user population size that is realistic for contemporary web applications), and for user modeling with intermittent network connectivity such as in user-adaptive mobile systems. In Section 5, we describe a simulation experiment to test the performance and scalability of our user modeling server in real-world application scenarios. We propose a theory of how such experiments should be conducted, based on available client-side usage data (and not server logs that have been exclusively used so far). Section 6, finally, summarizes the contributions and the resulting conclusions.

2 Traditional Data Repositories of Generic User Modeling Systems[1]

This section gives an overview of how user models have been stored in major generic user modeling systems that have been developed to date.

GUMS [Finin and Drager 1986; Finin 1989] allowed programmers of user-adaptive systems the definition of simple stereotype hierarchies and, for each stereotype, of Prolog clauses describing stereotype members and rules prescribing the system's reasoning about them. The system was implemented in CProlog [Pereira 1996]. A 'flat' set of Prolog clauses contained all assertions about the current user and stereotypes, as well as definitions of all Prolog predicates that prescribed the system's inference processes. In the spirit of Prolog, the authors refer to the set of these clauses as a 'data base'. They were all maintained in Prolog's in-core, native store of clauses though, and the system's assumptions about the user were not separated from its other clauses.

BGP-MS [Kobsa 1990; Kobsa and Pohl 1995; Pohl 1998] allowed assumptions about the user and stereotypical assumptions about user groups to be represented in a first-order predicate logic. A subset of these assumptions could be stored in a terminological logic. Inferences across different assumption types (i.e., types of modals) could be defined in a first-order modal logic. All knowledge of the system was encoded in a 'flat' list of clauses for the theorem prover OTTER [McCune 1994], and was read into OTTER from one or more input files. Alternatively, terminological knowledge and instantiated ground clauses could be expressed in a KL-ONE-like language [Kobsa 1991]. All system knowledge including all user models were maintained in the ASCON network storage [Bosch 1988] which was fairly efficient due to extensive hashing, and stored persistently in a single file.

PROTUM [Vergara 1994] was implemented in IF Prolog and represented user model content as a list of constants, each with associated type (i.e., observed, derived from stereotype, default) and confidence factor. User models were stored in separate files per user and could also be imported from the BGP-MS user model repository. They were kept in main memory during runtime.

UMT [Brajnik and Tasso 1994] allowed the user model developer the definition of hierarchically ordered user stereotypes, and of rules for user model inferences and contradiction detection. UMT models were objects within the memory space of the Lisp process, and were collectively read from and saved to a text file.

TAGUS [Paiva and Self 1994; Paiva and Self 1995] represented assumptions about the user in first-order formulas, with meta-operators expressing the assumption types. It allowed for the definition of a stereotype hierarchy, a library of misconceptions, and a number of user models. The system was programmed in Prolog and Gödel [Hill and Lloyd 1993]. Each user model was stored in a separate file that could be loaded anytime during runtime, and was maintained in main memory during execution.

um [Kay 1995] was a user modeling toolkit that represented assumptions about users' knowledge, beliefs, preferences and other characteristics as attribute-value pairs. Each piece

[1] The authors would like to thank Guido Bosch, Giorgio Brajnik, Tim Finin, Judy Kay, Joseph Konstan, Ana Paiva, and John Riedl for clarifications regarding the storage of user models in their respective user modeling systems.

of information (aka 'component') was accompanied by an indication of its source, a list of evidence for its truth and falsehood, and other data. Partial models kept related components together and organized them into hierarchies. The system was written in C and used the UNIX directory system to store and organize the user models, with each partial model being a directory. Standard UNIX permissions were used for access control. The persistent form of a user model was stored in text files that were intended to be readable by users. More recent user modeling servers from the same lab, Personis [Kay et al. 2002] and PersonisLite [Carmichael et al. 2005], store components and evidence in two separate databases. They were stored in Berkeley DB, which is not fully relational but very fast for large databases.

DOPPELGÄNGER [Orwant 1993, 1994, 1995] was a user modeling server that accepted information about users from hardware and software sensors, collected them in user models that were stored on the server, and allowed learning algorithms to operate upon the sensor data and upon user models. Each user model contained a primary model, submodels and backup models. At the storage level, a user model was a UNIX directory, and its components were files in this directory. In fact, DOPPELGÄNGER was based on an early version of um (described in [Kay 1990]). The decision to use many files was made to allow many processes the access to a given user model at the same time [Orwant 1993].

Group Lens [Tornago 2006] originally employed various collaborative filtering algorithms [Breese et al. 1998; Herlocker et al. 1999] for predicting users' interests, based on explicitly provided users ratings, implicit ratings derived from users' navigation, and transaction histories (e.g., shopping basket operations, purchases). GroupLens stored all user ratings in a database, but kept a correlation matrix of all ratings in cache memory during runtime. This created memory problems and huge performance problems on the largest sites. They were solved temporarily with reduced-size models being selected statistically (with careful sampling, the reduced-size models did not show much quality degradation). Group Lens eventually moved to item-item models, which can be truncated substantially without much loss of quality [Miller et al. 2004].

With the exception of Group Lens and, recently, Personis [Kay et al. 2002] and PersonisLite [Carmichael et al. 2005], none of the developed generic user modeling systems seemingly paid much attention to appropriate storage mechanisms. In the simplest case, all user models were read from secondary storage at launch time (or were already part of the program code of the user modeling system in the first place). In the most sophisticated case, the model of the current user was individually read from a file at the beginning of the session with the respective user, and saved to a file thereafter. During a session with a user, the complete user model was maintained in main memory. In many cases, user models were also tightly intertwined with the programming language (e.g., part of the LISP or PROLOG space). It is obvious that this approach does not scale up when the number of users increases.

This disregard of storage considerations is not surprising though since these generic systems were either designed for single-user applications only, or never tested with a larger number of parallel applications and users. Researchers regarded other properties of generic user modeling systems as more important, such as generality including domain independence, representational expressiveness, and inferential capabilities (see [Kobsa 2001]).

In the next section, we will propose directory systems as an alternative, which offer numerous advantages over previous approaches: demonstrated performance and scalability, dynamic and transparent management of distributed information, built-in replication and synchronization, a rich number of pre-defined types of user information, international vendor-independent standardization, and extensibility of the core representation language for new information types and for data types with associated semantics. Directories also allow for the virtual centralization of distributed user models and their selective centralized replication if better performance is needed.

3 Directories as the Foundation of User Modeling Servers

3.1 Introduction

Directories are specialized database management systems that were originally designed for maintaining information about people in organizations, and later extended to also include information about devices and services on a network. They are crafted to meet the needs of a wide range of applications and are based on international standards that guarantee interoperability between implementations of different developers and vendors.

The first of these international standards promulgated by ITU-T and ISO in the late 1980s was the Directory Access Protocol (DAP, [Chadwick 1996; ITU-T 2001]). DAP was intended to be used by clients for accessing an X.500 directory service. It did not gain much popularity, mainly because it was too complex to be implemented and deployed on the hardware that was typical for that time.

During the following years, LDAP emerged from the X.500 protocol family as a light-weighted alternative. LDAP removed excessive complexity from X.500 DAP, significantly reduced resource requirements, and took advantage of the popular TCP/IP rather than the OSI protocol stack. At the same time LDAP still preserved many strengths of X.500, including its information model (see [Fink 2004]), its versatility, and its openness. Many commercial systems have been developed that are largely LDAP-compliant, including network-wide address books (e.g., Lotus Notes[IBM 2006a] and Microsoft Exchange [Microsoft 2006a]), network operating system directories (e.g., Microsoft Active Directory [Microsoft 2006b], Novell eDirectory [Novell 2006], IBM Tivoli Directory Server [IBM 2006b], and Sun Java System Directory Server [Sun 2006]), and special-purpose Internet directories (e.g., [Bigfoot 2006; Switchboard 2006; WhitePages.com 2006]).

In the following, we will briefly 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, including the historical development of directories, we refer to [Howes and Smith 1997; Howes et al. 1999; Loshin 2000].

3.1.1 Information Model

The basic unit of information in an LDAP directory is an *entry*. An entry represents information about a (real-world) object, specifically a person, an organizational unit, a resource, or a service. Below is an example of a directory entry for the hypothetical user Peter Smith (in LDIF, the LDAP Data Interchange Format). In this example, `dn` is the 'distinguished name' that will be explained further below. The entry is related to the object classes `top` and `person`, and has the attributes `cn` (for common name), `sn` (for surname), `age`, `sex`, and `continent`.

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

Each directory entry belongs to one or more *object classes* (e.g., `person`, `device`, `organizationalUnit`). Object class definitions specify a class type, a set of mandatory and a set of optional attribute types, and an object identifier. Each attribute is associated with an attribute type, whose definition specifies a name and an object identifier, an indicator whether one or multiple values are allowed for an attribute, an attribute syntax, a set of matching rules that specify how attribute values are to be compared for equality, ordering, and substring matching, an indicator whether an attribute is intended to be used by the system or the user, and possible restrictions on the range or size of attribute values.

3.1.2 Naming Model

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

In a hierarchical file system, the root directory is the common ancestor and contains all files and directories of the hierarchy. The root entry of an LDAP tree, in contrast, is a special entry that contains server-specific information (e.g., about supported LDAP versions, available operations and security features, and backup servers that can be contacted in case of a breakdown). No domain data can be placed in the root entry of an LDAP tree. In a hierarchical file system, a node is either a file or a directory, but not both. In contrast, entries in an LDAP tree contain data (represented as attribute-value pairs) and at the same time can have child entries underneath them.

The final difference between LDAP and a hierarchical file system relates to the naming of individual nodes within the tree through full path specification. Names in LDAP are in reverse order compared with, e.g., the UNIX name convention (i.e., the leaf entry comes first). The distinguished name of our above example entry,

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

is formed by concatenating the comma-separated names of the entries from the leaf to the root (UMS thereby stands for the entry 'user modeling system', and `gmd.de` for the organization that hosts it).

LDAP also supports non-hierarchical topographies by so-called 'alias entries'. Using the analogy to the UNIX file system again, aliases are comparable to symbolic links. Although aliases can be used for connecting directory partitions that reside on different LDAP servers, LDAP's facility of choice for 'intra-linking' distributed directories are '*referrals*'. 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 1). The main advantage of referrals is that they are standardized as a part of the LDAP v3 specification.

3.1.3 Functional Model

The LDAP functional model comprises three groups of operations for accessing a directory:

- *query operations* (namely *search* and *compare*) allow for searching and retrieving information,
- *update operations* allow for adding, deleting, renaming, and modifying entries,
- *authentication operations* (namely *bind* and *unbind*) *and control operations* (*abandon*) allow for authenticating clients and servers and for controlling previously initiated LDAP operations.

The server verifies a client's credentials and, if approved, grants it certain access privileges. These privileges persist until the end of a session or until the client re-authenticates. Clients can terminate a session anytime using *unbind*, and terminate ongoing LDAP operations (e.g., a long-running search) using *abandon*. Besides these predefined operations, custom operations can also be defined in a standardized manner by taking advantage of the '*extended operation*' facility.

3.1.4 Security Model

LDAP's security model provides standardized support for *authentication, signing,* and *encryption*:

- *Authentication* allows for the verification of the identity of another party. LDAP's security model offers a standardized interface to various authentication schemes including anonymous authentication (i.e., no authentication), simple passwords, (communicated as plain text or encrypted via an SSL-secured connection), X.509 certificate authentication via SSL, and SASL-based authentication and encryption using e.g. Kerberos.
- *Signing* ensures the authenticity and integrity of information exchanged between clients and servers. LDAP's security model supports signing through e.g. SSL. Within an SSL connection, each block of information is accompanied by a cryptographic checksum that allows clients and servers to verify the sender and to check whether the data has been tampered with during transit.
- *Encryption* allows for the encoding of all exchanged information. During the negotiation phase of an SSL connection, the two parties (e.g., a client and a server) agree on a protocol (e.g., RC4, DES, IDEA). Besides SSL's encryption facilities, LDAP's security model also supports alternative encryption services (e.g. MD-5) via its SASL interface.

Against this backdrop, it may come as a surprise that there is currently no standard access control mechanism for LDAP. In our work, we decided to take advantage of the access control model offered by Sun Java System Directory Server for granting directory access to anonymous and authenticated clients. Directory Server establishes access control through a set of access control lists, each of which implements an access control rule and is usually attached to a directory entry via the special attribute `aci` (for access control information). By default, all users are denied access of any kind to the directory. An `aci` then grants or denies access to its directory entry and to all entries beneath. Its granularity can be very fine, if necessary down to an operation type on a single attribute of a single node issued by a particular user from a dedicated IP address during a specific period in time.

3.2 Directories versus Databases

In the following, we identify six characteristics of user modeling servers that seem to be important for the effective support of user-adaptive applications: the availability of pre-defined schemas that are relevant for user modeling and can be freely extended; the management of distributed information, replication, performance and scalability, adherence to open standards, and consistency management. We argue that directory systems generally rate higher than traditional database systems on the first five of these dimensions. While database systems rate better with regard to consistency management, there currently seems no user modeling application around that would impose such requirements. We also discuss the ability of LDAP to not only support centralized but also 'virtually centralized' user modeling servers, which we regard as crucial for successful deployment to practice. This is not an independent characteristic though, but a consequence of the other properties that we discuss. We finally conclude that the wealth of useful characteristics for user modeling should make directories, and specifically LDAP, the storage of choice for user modeling servers. Databases should only have a role in smaller applications where these characteristics are not so important, and in applications that pose high demands on consistency management.

3.2.1 Pre-defined Schemas and their Extensibility

LDAP directories provide built-in support for storing and retrieving various kinds of people-related information including names, phone numbers, salaries, photographs, digital certificates, passwords, preferences, and even mobile 'user agents'. Moreover, they support the representation of information about organizations, groups (e.g. administrators) and devices (e.g. printers). Pre-defined schemas exist for these information types (e.g., `organization`, `organizationalUnit`, `device`, `person`, `residentialPerson`, `organizationalPerson`, `organizationalRole`, etc.).

Directories are not limited to a fixed schema though: 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 not only includes new types of information (e.g., descriptions of user modeling services, users' locale), but also custom primitive data types with new semantics (e.g., German telephone numbers, probabilities of users' interests) and behavior (e.g. dynamic entries [Yaacovi et al. 1999], such as transient information about the user's locale that must be periodically refreshed in the user model).

A set of pre-defined primitives for representing basic information about users is surely advantageous, specifically if they are standards-based (see Section 3.2.5). Such primitives facilitate the exchange of information between applications and user modeling servers, and between different user modeling servers. They may also speed up the development of user modeling servers in general. Likewise, ease of extensibility is also crucial since there currently exists no generally accepted user model ontology yet (see [PAPI 2001; Razmerita et al. 2003; Heckmann et al. 2005] for rudimentary beginnings though). When comparing LDAP with current database systems, LDAP clearly possesses far more predefined types of user information and methods for extending them. In contrast, only few database systems offer user-related information types and functionality for defining new primitive data types via low-level extensions to the database nucleus (e.g. [Informix 2006]). These features are proprietary though and therefore impair the interoperability between different database systems and their clients.

3.2.2 Management of Distributed Information

LDAP directories can manage information that is dispersed across a network of servers by linking this information through referrals. In the example of Figure 1, the user models stored on server B and the Usage Model stored on server C can still be accessed from server A since it is linked to the administrative structure on A (for a brief presentation of the User Model and the Usage Model, we refer to Chapters 4.2.1.1 and 4.2.1.2). Historically, this feature has been utilized for deployments where the responsibility and authority for the management of directory information is distributed (e.g., a branch of a firm is responsible for information about local employees). The distribution is transparent to the outside world, i.e. the directory appears as a single consistent repository. Scalability considerations provide another important motivation for distributing information. It is often better to design a large directory as a network of smaller parts, since this often guarantees much better performance, scalability, availability, and reliability of the overall service than a single large directory. Moreover, a distributed directory is in many cases cheaper to implement and simpler to manage (see [Howes et al. 1999] for more information on distributed LDAP deployments and resulting advantages).

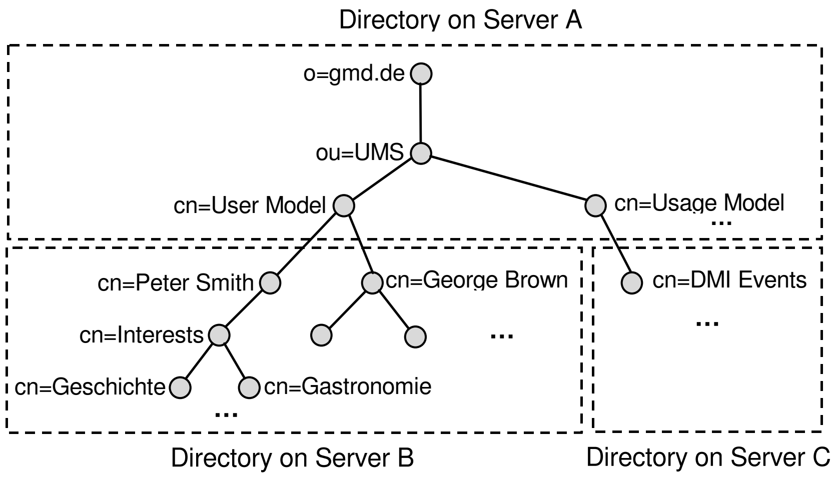


Figure 1: Distributed directory (based on [Howes et al. 1999])

The ability to maintain distributed repositories with user information already seems important today. It will become even more important in future applications of user modeling servers, for the following reasons:

- *User information is already distributed in current applications.* Customer relationship management systems on the web utilize several sources of user information, such as user profiles, purchase records from legacy systems, and customer segmentations from marketing research. Commercial user modeling servers that support such applications integrate these information sources already today to a greater or lesser extent [Fink and Kobsa 2000].
- *Distribution is foreseen for identity management on the web.* Traditional identity management systems (e.g. [Passport 2006; Yodlee 2006]) require that users store their data in a single web repository. More recent developments such as [Liberty 2006], [enQuire 2006] and [Kobsa and Schreck 2003] pursue a federated approach, which ensures "that the use of critical personal information is managed and distributed by the appropriate parties, rather than a central authority" [Liberty 2006]. Federated identity management carries fewer security risks since it avoids a single point of failure, caters better to users' privacy concerns by not forcing them to divulge all data to a single authority, and also gives the local repositories more control over the recipients of user information.
- *User modeling and user models are becoming ubiquitous.* User models for smart appliances have already appeared on the market or will become reality soon that maintain relevant user characteristics (e.g., interests and preferences) and adapt their functionality accordingly. Examples of such appliances include
  - car radios that learn drivers' preferred stations, volume and tone, and whether to interrupt with traffic alerts, and store their preferences on their personal car keys,
  - mobile phones that pre-load web pages that are presumably relevant (e.g., stock quotes),
  - DVD and video recorders that proactively record television programs that are presumably interesting to a TV viewer according to the preferences that it learned from their viewing patterns, and
  - refrigerators that track the stored food and reorder out-of-stock items via the Internet, thereby taking a user's preferences into account.

  While these small user-model applications are largely independent of each other, it may make sense in some cases to let them communicate with each other and exchange user information, such as a wristwatch contacting the refrigerator while in a grocery store and reminding the user to buy groceries that are running low.

Database systems can handle data distribution too. The possible scale and granularity of distribution, however, are 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. An extension of LDAP, the Connection-less Lightweight Directory Access Protocol (CLDAP, [Young 1995]) additionally facilitates the quick look-up of attribute values without the need for a permanent connection, which becomes very interesting in mobile scenarios and for appliances with limited capabilities.

3.2.3 Replication

Replication of user model information is very attractive for remote user-adaptive applications. It allows these applications to temporarily duplicate the whole user model or substructures, to manage them locally while offline (including local edits and additions), and to synchronize the local copies with the original data (which may have also changed in the meantime) when this becomes possible again. Figure 2 shows a situation where the User Model on server A has been replicated in its entirety both on servers B and C (we omitted the labels in the replicas for reasons of brevity).

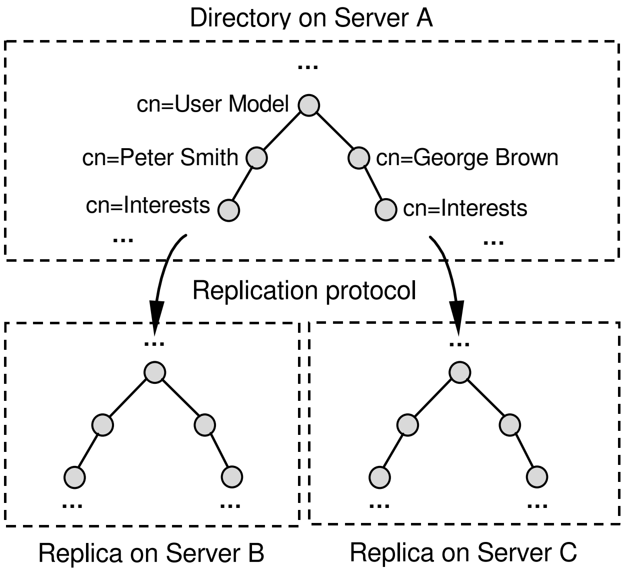


Figure 2: Replicated User Model (based on [Howes et al. 1999])

Historically, replication was primarily motivated by availability and performance considerations when deploying directories to real-world environments. Maintaining replicas of directory information can significantly increase the availability and performance of a directory service from a client's point of view. These benefits can be equally leveraged in user modeling scenarios:

- *Availability*. If a remote user modeling server, or the network connecting it with a user-adaptive application, becomes (temporarily) unavailable, access to a local replica of user information enables the application to still provide personalized information and services. This can e.g. increase the autonomy of mobile users, smart appliances and user agents, by reducing their dependence from the availability of a network connection.
- *Locality*. In general, the closer user information is to client applications, the better the quality of service and, in some cases, the achievable level of security. Creating a local replica of a user model may increase the security of user information since network communication can be reduced to what is necessary to keep replicas synchronized (see [Kobsa and Schreck 2003] for related threat models).

- *Performance*. Replicating a central repository on a network may also increase performance. For instance, user-adaptive applications can avoid network congestion by utilizing a local replica of a user model rather than one that is retrieved from a remote user modeling server.

Replication is currently far more powerful in directory systems than in database systems. For one, LDAP directories are replicated on a far larger scale than databases. For example, employee directories in international organizations may have replicas in hundreds or thousands of branch offices all over the world. In contrast, only few database systems support replication, and then typically with few copies only. It should be noted though that the replication and distribution mechanisms of LDAP may also create disadvantages, such as temporary inconsistencies between a user model and its copies, and the computing efforts needed to resolve such replication conflicts. Furthermore, directory replication requires careful planning and deployment (see [Howes et al. 1999] for a detailed discussion).

3.2.4 Performance and Scalability

Performance and scalability are very important criteria for user modeling applications, particularly when those are to be deployed to the web. Directories have been designed to meet the needs of a wide variety of Internet and intranet applications (e.g., e-mail servers and clients, Web server applications and browsers, groupware servers and clients, and lightweight database applications). Scalability is therefore of paramount importance for directories since the number of applications that will utilize them is often not known at the time of deployment. Databases, in contrast, are often designed for a dedicated set of database applications.

Directories are specifically optimized for search operations; their performance with regard to updates is considered less important. While it is true that performance is vital for database systems as well, their typical workload differs considerably from directories. Databases are optimized for a relatively balanced ratio of search and update operations (as is the case for many commercial transactions). [Shukla and Deshpande 2000] found that databases outperform directories when a given query matches a large number of database entries or has a large result set. If the number of matching entries and the overall result set are small, however, directories showed a far better search performance in the authors' evaluation than databases.

3.2.5 Open Standards

Adherence to open standards in the design of user modeling clients and servers is very important since this improves their interoperability. There already exist efforts in some subfields of user modeling to come up with standards (e.g., [Kobsa et al. 1996], [Kummerfeld and Kay 1997], [Goodman et al. 1999], and [LTSC 2006]), although without many results so far.

LDAP is highly standardized. The most important areas of standardization are:

- *LDAP protocol specifications* for versions 2 and 3, namely RFCs[2] 1777-1779 and 2251-2256. Some of these standards are in turn based on X.500 standards (e.g., RFC 2256 which defines the syntax and matching rules for attribute types and object classes in an LDAP user schema, is based on X.501, X.520, and X.521).
- *Proposed extensions to LDAP version 3,* including RFC 2589 for managing dynamic LDAP entries that need to be periodically refreshed by client applications in order to persist; RFC 2820 for common requirements towards interoperable LDAP access control models; RFC 2713 for LDAP schema elements that represent Java objects; and RFC 2714 for schema elements that host CORBA object references.
- *Related Internet standards or proposed standards* that have been adopted by LDAP, including the Simple Authentication and Security Layer (SASL, RFC 2222). SASL is a generic framework for negotiating security parameters between applications, e.g., for authentication, encryption and signing. LDAP version 3 provides native support for SASL.
- *Additional security standards* besides SASL, such as X.509 certificates (RFCs 2559 and RFC 2587), the Secure Sockets Layer protocol (SSL) and the Transport Layer Security protocol (TLS, RFC 2246).
- *The LDAP Data Interchange Format* (LDIF, RFC 2849), a text-based format for representing and exchanging directory content.
- *A C programming interface* for LDAP (RFC 1823), and several APIs for Java that are available as Internet Drafts (e.g. [Weltman et al. 2005]). Moreover, there exist a number of proprietary and mostly freely available software development kits (SDKs) for a variety of languages including C, C++, Java, Perl, and Basic.

Although several standards exist for databases as well (e.g. [ISO 1989, 2003] for SQL), their number and scope falls far short of those for directories. This lack of standardization has many implications, most importantly that no real interoperability can be achieved between database systems of different vendors (e.g., an Oracle client application will generally not work with a Sybase database).

3.2.6 Transaction and Replication Consistency

A transaction is a group of logically coherent operations, e.g. a set of queries and additions that result in a low-level adaptation at the interface (including all internal inferences in the user model, such as stereotype activation or de-activation). Transactions should adhere to the well-known ACID properties (*A*tomicity, *C*onsistency, *I*solation, and *D*urability; see [Fink 2004] for more information). Consistency, for example, 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., since integrity constraints are being violated), the user model has to be reset to the original state (see Fink [Fink 1999] for related examples).

[2] RFCs (Request for Comments) are documents of the Internet Technology Task Force (http://www.ietf.org). They describe many aspects of Internet communication, e.g., networking protocols, procedures, programs, and architectural concepts.

To the best of our knowledge, none of the interfaces and communication protocols that have been proposed in the user modeling literature so far (e.g., [Kay 1995; Kobsa and Pohl 1995; Orwant 1995; Paiva and Self 1995; Kobsa et al. 1996; Carmichael et al. 2005]) puts ACID-compliant transactional facilities at the disposal of the application developer. A static and restricted form of transaction support can be found in a number of systems including the PAT-InterBook system [Brusilovsky et al. 1997] and in LDAP. Only a few directories support transactional consistency that goes beyond the scope of a single LDAP operation. Compared to this, database systems outperform directories with their full support for transactions.

Database systems and directory systems also exhibit differences with regard to replication consistency. Directory systems support hundreds or thousands of replicas (e.g., a globally distributed staff directory of a multi-national firm). Deployed user modeling currently seems to be implemented with an assumption that loose consistency is acceptable. The administrative overhead is therefore kept low. Databases in contrast normally support strong consistency, i.e. database replicas have to be in sync at all times. Maintaining such strong consistency, however, requires a considerable amount of system resources (see [Fink 2004]). This is one of the main reasons why databases normally support a small number of replicas only.

User modeling currently seems to be content with loose consistency. We are e.g. not aware that user-adaptive web stores that record in a user profile the items that the user put into her shopping cart see a strong need to correct such an entry in the unlikely event that the request to the remote shopping cart server fails or times out. Likewise, developers of user-adaptive handheld guides that foresee local replication of a user model on the Internet to respond to temporary connection failures do not seem to see a pressing need to update the local user model immediately when the central model changes (e.g., since the user accesses information on a public terminal rather than the handheld device), and vice versa.

3.3 Virtual Centralization Using LDAP

Integrating user information that is scattered across a network (no matter whether it refers to the same or different users) is of paramount importance, specifically for businesses [Fink and Kobsa 2000]. There has been some debate recently whether this integration should occur proactively by storing all user information in a central repository, or ‘on the fly’ when needed (specifically through communicating agents, each of which is in charge of one of the local repositories that need to be integrated). [Fink and Kobsa 2000] emphasize the merits of the centralized approach, which include: up-to-dateness of user information; avoidance of duplication and resulting potential for inconsistencies; compact storage when generic classes (‘stereotypes’) are present; easy availability to different applications (and possible synergy effects due to the fact that user information acquired by one application can be employed by other applications and vice versa); and increased security [Schreck 2003].

All current commercial user modeling systems follow the centralized approach [Fink and Kobsa 2000]. [Yimam and Kobsa 2003] report that in a slightly different domain, namely expert finding in organizations based on locally stored expertise models, the performance of the agent-communication approach turned out to be unacceptable. They therefore argue for an aggregate expert model that is continuously fed from the individual local models.

Central servers in general also pose challenges, namely with regard to availability and scalability. [Orfali et al. 1994] discuss these problems in detail, and present solutions with regard to availability. Other reasons for avoiding centralization may also exist: [Vassileva et al. 2003] report having abandoned their originally centralized solution for a university-wide collection of student-related information since central processing no longer suited the heterogeneity of user information, usage purposes, and decision rules.

In their ‘future work’ section, [Fink and Kobsa 2000] already pointed out that ‘centralized user modeling’ does not necessarily imply the physical centralization of user information (although this has been the case in all research prototypes and commercial user modeling servers that have been developed so far). An alternative is the concept of ‘virtually centralized’ user information, which is extremely well supported by LDAP. Virtual centralization can come in many shades. Figure 1 shows a situation in which A is the central access point to all user information on the server. Integration of information on B and C is achieved by following links at runtime.[3] If runtime integration is detrimental to performance, or not advisable because of insufficient availability of B and C, some or all information on B and C can be replicated on A.

An interesting aspect is that B and C do not even need to be LDAP servers to allow for virtual integration. [Fink 2004] explains in detail how directory synchronization software from vendors like [Persistent 2006] and [Critical Path 2006] allows for a fusion of user data in legacy systems with an LDAP user modeling server. This can be achieved through a meta directory that contains a replica of the data in B and C. Integration of non-LDAP data can however also be realized through a virtual directory that retrieves values from legacy databases on demand.[4]

It should be noted that the support of virtual integration is not an independent property of LDAP, but rather a consequence of its support for distributed repositories, (loose) consistency management, and replication. In a virtually centralized user model architecture, furthermore, the decision between central storage, distributed storage, and distributed storage with (partial or full) central replication is not one that is made by an application developer who is in charge of the user modeling aspects. Instead, it will be made by a system administrator based on service quality and consistency needs. It may well be that much of the tension between centralized and distributed user models [Vassileva et al. 2003], while valid from a systems administration point of view, is a non-problem from a user modeling point of view.

3.4 Consequences for User Modeling

Directories generally surpass databases with regard to the availability of pre-defined schemas for people-related information, extensibility, the management of distributed information, the possible extent of replication, performance and scalability, and adherence to standards. User modeling servers and clients that take advantage of directory technology are likely to enjoy a considerable degree of openness and flexibility. As has been explained in Section 3.2.4 though, databases manage consistency better, both with regard to individual transactions and with regard to global consistency between the original data and their replicas. They also perform better when large amounts of data have to be retrieved, or when the number of updates approaches the number of seeks.

---

[3] [Fink 2004] discusses how link chasing can occur at different levels of transparency, with different security implications.

[4] Directory synchronization software allows for one-way and two-way synchronization of data that may be fused from several probably heterogeneous data sources (see [Fink 2004] for details).

Based on our discussion of the need for these characteristics in the area of user modeling, we argue that directories and not databases should be the storage mechanism of choice for user modeling servers. Databases should only have a role in smaller applications where the characteristics in which LDAP excels are not so important, and in applications that pose high demands on consistency management. To the best of our knowledge, however, there does not seem to be a strong demand in the current user modeling literature for a high data refresh rate in the user model, nor a need for retrieving large amounts of user information or for very sophisticated transaction mechanisms (see Fink [1999]).

[Kummerfeld and Kay 1997] already considered LDAP as a candidate for a user model access protocol, but dismissed it on the grounds that it is difficult to "extend the types of information stored in the database 'on the fly'". While the schema can be changed, "this is usually a job for the directory administrator and cannot be done easily by a user program during a session". The authors reason though this may not be a constraint for many applications.

This deficiency has since been addressed: LDAP version 3 now includes client-side schema manipulation options [Wahl et al. 1977]. In retrospect, the objections of [Kummerfeld and Kay 1997] have been vindicated. Schema extensibility became an indispensable feature for today's deployment scenarios, since it allows one to cater to evolving data requirements without affecting the overall service. With previous LDAP versions, extending the schema (e.g., adding new attributes to a user's profile) typically required an administrator to stop the LDAP server, change the schema, start a reconfiguration process, and finally reboot the server. Such a procedure is unacceptable in today's application environments, which require a service with basically no downtime.[5]

[5] According to IEEE, the lowest availability level called 'stable' is defined by 99% uptime, which translates into 3.7 days of planned or unplanned downtime per year. Most commercial directory servers, however, have to be operated at the service level of 'high availability', which mandates 99.99% uptime, i.e. a maximum downtime of 52.6 minutes per year.

4 Architecture of an LDAP-Based User Modeling Server

4.1 Overview

Following our conclusions in Section 3.4, we developed our user modeling server (UMS) as an LDAP directory server[6] that is complemented by several 'pluggable' user modeling components and can be accessed by external clients. Figure 3 gives an overview of this architecture. The central *Directory Component* comprises the sub-systems *Communication*, *Representation*, and *Scheduler*. The Communication sub-system is responsible for handling the communication with external clients of the user modeling server (e.g., user-adaptive applications), and with the User Modeling Components which are internal clients of the Directory Component. Each User Modeling Component performs a dedicated user modeling task, such as collaborative filtering, domain-based inferences, etc. The Representation sub-system is in charge of managing the directory contents (mostly user information). The main tasks of the Scheduler are to wrap the LDAP server in a component interface and to mediate between the different sub-systems and components of the User Modeling Server.

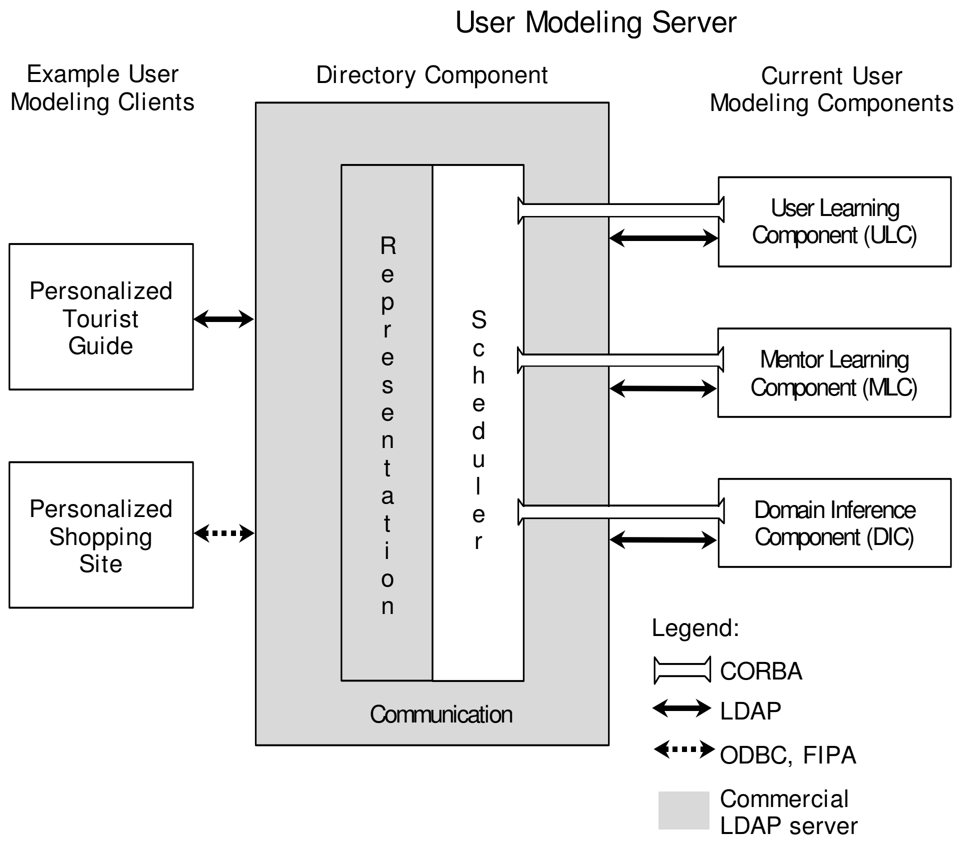

User Modeling Server
Example User Modeling Clients
Directory Component
Current User Modeling Components
Personalized Tourist Guide
Personalized Shopping Site
Representation
Scheduler
Communication
User Learning Component (ULC)
Mentor Learning Component (MLC)
Domain Inference Component (DIC)
Legend:
CORBA
LDAP
ODBC, FIPA
Commercial LDAP server

Figure 3: Overview of the server architecture

[6] We used the Directory Server from iPlanet which has meanwhile morphed into Sun's Java System Directory Server [Sun 2006].

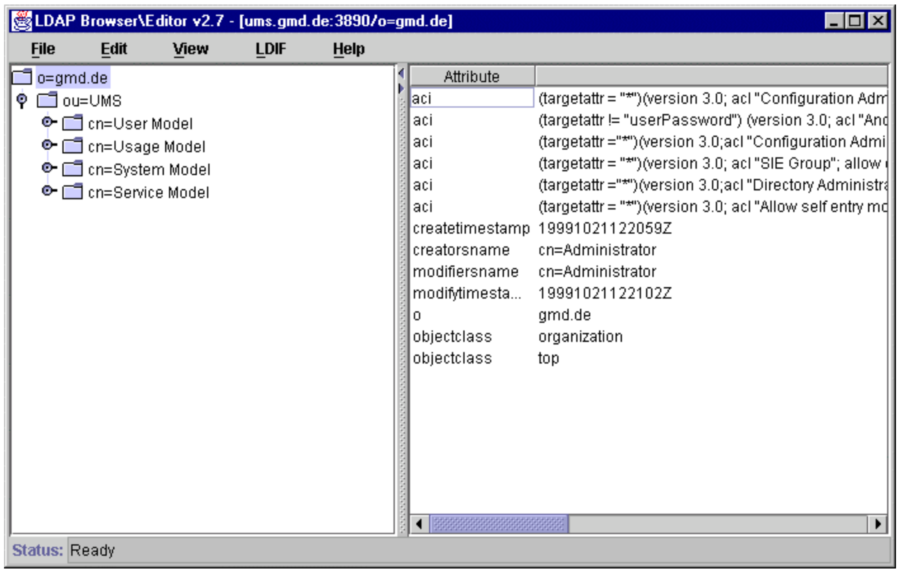
The Directory Component and the User Modeling Components communicate via CORBA [Pope 1997; OMG 2001] and LDAP. This communication infrastructure does not mandate a specific distribution topography. Components can be flexibly distributed across a network of computers, e.g. dependent on available computing resources.

Below we describe the Directory Component in more detail. The examples will be taken from a deployment of our user modeling server in the Deep Map project [Malaka and Zipf 2000; Deep Map 2001], which is concerned with the development of personal web-based and mobile tourist guides. We then summarize the User Modeling Components and the external User Modeling Clients that were defined in this project (see [Fink and Kobsa 2002; Fink 2004] for a more detailed description). While the individual elements of the user modeling server can be selected and configured differently for each application scenario, the choices that were made in the Deep Map project are very representative of web-based applications that cater to users' interests.

4.2 Directory Component

4.2.1 Representation Subsystem

The task of the Representation subsystem of the Directory Component is to store various models. 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. When adding new object classes to 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 shown in Figure 4.



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

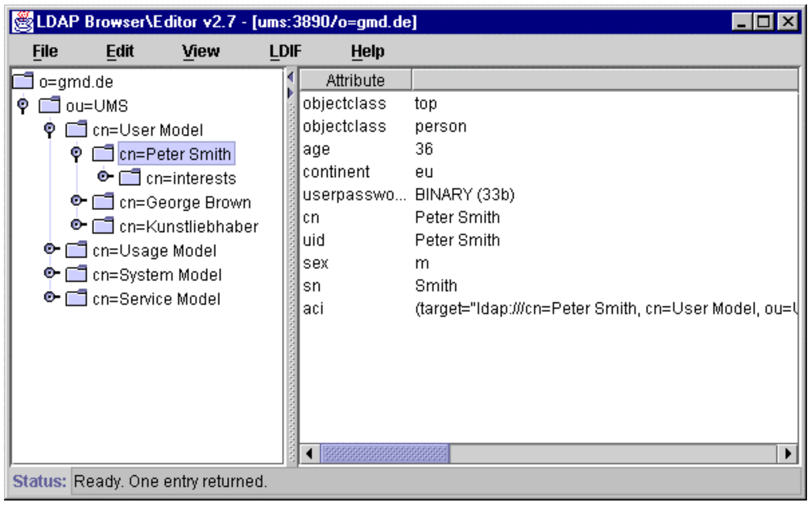
Attribute	
aci	(targetattr = "*")(version 3.0; acl "Configuration Adm
aci	(targetattr != "userPassword") (version 3.0; acl "And
aci	(targetattr ="*")(version 3.0;acl "Configuration Admi
aci	(targetattr = "*")(version 3.0; acl "SIE Group"; allow
aci	(targetattr ="*")(version 3.0;acl "Directory Administra
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 4: Overview of the models hosted by the User Modeling Server

In the right frame of the browser, we see various attributes and associated values for the currently selected root entry `gmd.de`. One important attribute is `aci`, which provides access control information. Others are the name (`gmd.de`) and the standard LDAP object classes that are associated with the root entry (namely `top` and `organization`). Each of these object classes adds a number of required and optional attributes. The class `organization`, for instance, adds name, postal address, telephone number, etc. (only those attributes of `gmd.de` that contain at least one value are shown in Figure 4). All mentioned attributes may be modified by clients provided they have sufficient access rights. The so-called ‘operational attributes’ `createtimestamp`, `creatorsname`, `modifiersname` and `modifytimestamp` record all modifications and may not be changed. For more information on LDAP’s Information Model and Naming Model, we refer to Sections 3.1.1 and 3.1.2 as well as Chapter 5 of [Fink 2004].

4.2.1.1 User Model

The left frame of Figure 5 shows three user models, one for `Peter Smith`, one for `George Brown`, and one for a stereotype `Kunstliebhaber` (art lover). In general, user models in Deep Map 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` (whose entry is currently selected in the left frame) are shown in the right frame of Figure 5. Since `Peter Smith` was assigned to the object classes `top` and `person`, the demographic part comprises required attributes of `person` (namely the common name `cn` and the surname `sn`) and optional attributes (e.g., the encrypted `userpassword`). Other visible application-specific attributes include `age`, `continent` and `sex`. Several other inherited attributes (e.g., `description`, `telephone number`) have not been filled with values yet.



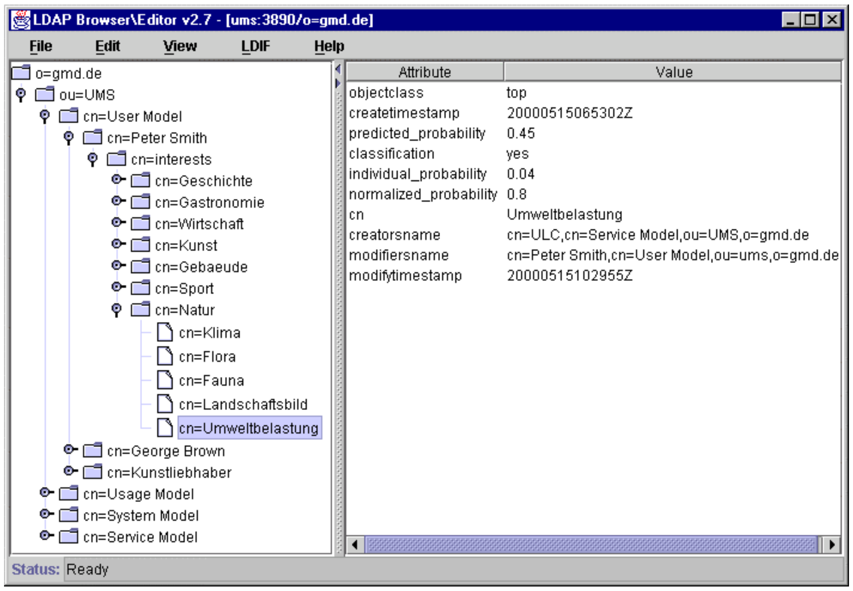
```
o=gmd.de
  ou=UMS
    cn=User Model
      cn=Peter Smith
        cn=interests
      cn=George Brown
      cn=Kunstliebhaber
    cn=Usage Model
    cn=System Model
    cn=Service Model
```

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=

Status: Ready. One entry returned.

Figure 5: User models

The major part of a user model in Deep Map describes users' interests and preferences. The topography and terminology of this part corresponds to the domain taxonomy of Deep Map[7], which is maintained in the System Model (see Section 4.2.1.3). Figure 6 depicts the user model of Peter Smith, with his `interests` being unfolded in the left frame. They range from `Geschichte` (history) to `Natur` (nature), which itself is divided into several sub-interests.



```
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

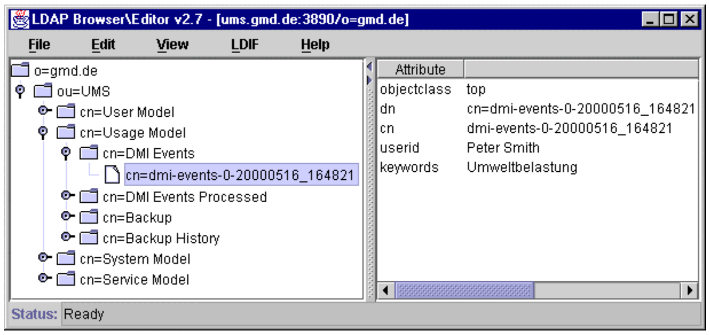
Figure 6: Interest model of Peter Smith

The interest in `Umweltbelastung` (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. They mostly represent inferences and probabilities that are being computed by the User Modeling Components in the Deep Map domain (see [Fink and Kobsa 2002] for detailed explanations).

4.2.1.2 Usage Model

The Usage Model acts as a persistent storage for usage-related data within the UMS. It comprises usage data communicated by the application, 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`). In the left frame of Figure 7, we see the hierarchy of the Usage Model from an administrator's point of view. It comprises the following parts:

[7] This correspondence can be weakened or even abandoned in deployment scenarios where a domain taxonomy cannot be defined beforehand (i.e., when an open corpus of terms is used). [Fink 2004] describes the configuration parameters that are available for such cases.



```
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
```

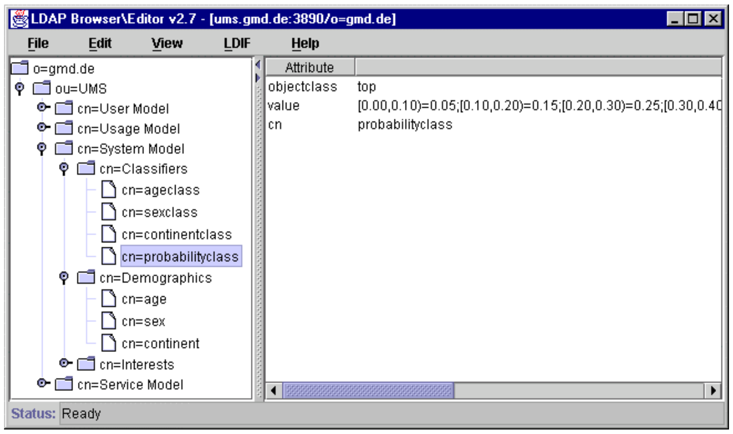
Attribute	
objectclass	top
dn	cn=dmi-events-0-20000516_164821
cn	dmi-events-0-20000516_164821
userid	Peter Smith
keywords	Umweltbelastung

Figure 7: Usage model

- `DMI Events` contains usage data communicated by the application. Each entry in this subtree describes a Deep Map 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 may be described through an attributed interest `Umweltbelastung`.
- `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 later processing and analysis, e.g. with visualization and data mining tools (which would be external User Modeling Clients in Figure 3).

4.2.1.3 System Model

The System Model includes relevant information about the application domain for User Modeling Components of the UMS, as well as other information that facilitates their operation. Its most important content is the aforementioned domain taxonomy.



```
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: System model: classifiers and demographics

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

- `Classifiers` contains templates for assigning continuous attribute values to classes.
- `Demographics` specifies those attributes in the demographic part of a user model that can be used for computing groups of similar users. In the current version of the UMS for Deep Map, this information is mainly relevant for the Mentor Learning Component (see Section 4.3).
- `Interests` constitutes the domain taxonomy of Deep Map. This sub-tree comprises five levels with nearly 500 leaf entries.

4.2.1.4 Service Model

The Service Model is divided into three parts, each of which is dedicated to a single User Modeling Component. Each entry represents a description of a server-internal event type in which a User Modeling Component is interested. So-called 'basefilters' allow one to restrict the portion of the overall taxonomy that must be monitored (e.g., `DMI Events` only). Events can be triggered before and after an LDAP operation is executed by the server. Post-notifications allow a User Modeling Component to react on the outcome of an LDAP operation (e.g., start processing an interface event that has been added to `DMI Events`). Pre-notifications allow a user modeling component to be invoked beforehand (e.g., carrying out consistency checks on interface events that have been added to `DMI Events`).

4.2.2 Scheduler

The second subsystem of the Directory Component is the Scheduler. Its main task 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 Section 4.2.1.4). This approach limits the amount of communication, allows for the addition and removal of user modeling components at runtime, and for their dynamic distribution across a network of computers. Event vectors submitted by the Scheduler are entered into a separate queue before being periodically processed by the ULC. This reduces the amount of synchronous communication between the ULC and the Scheduler to a minimum.

A second task of the Scheduler is the provision of user modeling extensions to the LDAP protocol. For instance, if a new user model has to be created, several standard LDAP operations must be executed in a particular order: checking for an already existing model, establishing the basic topography of a new model, setting appropriate access rights, and 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.

4.2.3 Communication

The Directory Component and the User Modeling Components communicate via CORBA [Pope 1997; OMG 2001] and LDAP. In Figure 3, the CORBA Object Request Broker (ORB) is depicted on the right side of the Directory Component as a software bus that mediates between the Directory Component and the User Modeling Components[8]. The two orthogonal communication layers are used at runtime as follows:

- The CORBA-based software bus is used for the communication of events and associated data from the Directory Component to the User Modeling Components (e.g., an event 'interest x inserted into user model y'). Components can register filter instructions with the Directory Component in order to subscribe to specific events or types of events. From a theoretical point of view, this communication resembles a 'filtered broadcast', i.e. a combination of the standard paradigms 'filters' and 'broadcast algorithms' for process interaction in distributed programming environments (see [Andrews 1991]).
- LDAP is employed by User Modeling Components for accessing and manipulating information that is hosted by the Directory Component.

Components can be flexibly distributed across a network of computers, e.g. dependent on available computing resources. The separation of event handling and information access on different layers provides for maximum flexibility. It even allows one to, e.g., replace the LDAP-based information management with one that is based on SQL, while still preserving the CORBA-based communication layer.

The communication between external UMS clients and the Directory Component is through LDAP, and also through ODBC for external clients that are not LDAP enabled (see Figure 3).

4.3 User Modeling Components

User Modeling Components (see the right side of Figure 3) perform dedicated user modeling tasks and communicate with the Directory Server through LDAP and CORBA. Arbitrarily many User Modeling Components can be 'plugged' into the User Modeling Server. In the Deep Map application, three User Modeling Components were developed (see [Fink and Kobsa 2002; Fink 2004] for more information):

- The User Learning Component (ULC), which learns user interests and preferences from usage data, and updates individual user models. It uses univariate significance analysis with a confidence interval to determine whether a certain type of observation about a user is made significantly more or less often than in the population sample (see [Mitchell 1997; Pohl et al. 1999; Schwab and Pohl 1999]).
- The Mentor Learning Component (MLC), which predicts missing values in individual user models from models of similar users. It employs memory-based Spearman correlation for determining the proximity between users and various weighted prediction algorithms from the area of collaborative filtering (see [Herlocker et al. 1999]).

[8] In the current version of the UMS for Deep Map, we use the commercial ORB VisiBroker [Borland 2006].

- The Domain Inferences Component (DIC), which infers interests and preferences in individual user models by applying domain inferences to user information that was explicitly provided by users or implicitly inferred by the ULC and the MLC. To this end, it performs sideward and upward propagation in the interest hierarchy of the domain taxonomy (see Section 4.2.1.3 and [Kobsa et al. 1994]).

4.4 External Clients

External User Modeling Clients provide information about users to the Directory Component, and retrieve information about users from it. Examples for such clients include:

- User-adaptive applications, which submit observations about the user to the Directory Component, and query the Directory Component for user characteristics.
- LDAP browsers, which system administrators use to configure the components of the Directory System, to specify access rights, and to add, update and remove entries.
- Widely available LDAP enabled applications, which users can employ to inspect and edit their user models (e.g., Microsoft Internet Explorer, Qualcomm Eudora, Microsoft Active Directory Browser); and
- LDAP or ODBC enabled data mining and visualization tools, to analyze the total user population (e.g., to find clusters, stereotypes, and other regularities) and thereby to indirectly verify the accuracy of the employed user modeling methods.

4.5 An Interaction Example

To illustrate the interplay between external User Modeling Clients, the Directory Server, its Scheduler, and the User Modeling Components, we briefly describe a scenario in which a user-adaptive application identifies a user interest and enters it into the Directory (see Figure 9).

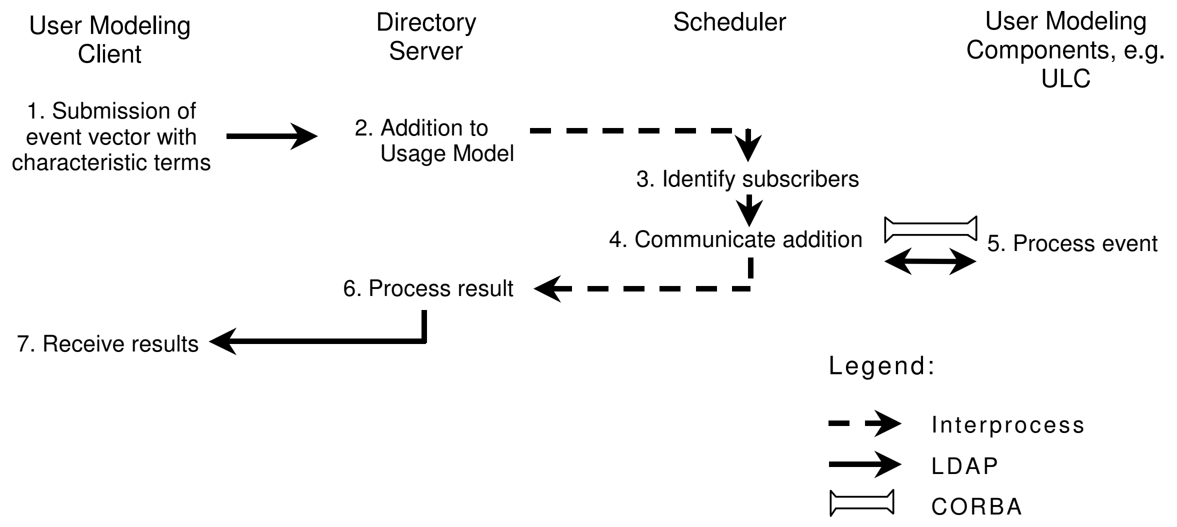


Figure 9: Interaction scenario

1. A user's request for a web document results in an event vector, which includes one or more terms that characterize the content of the document (the terms could come from the HTML 'description' and 'keywords' tags, or be selected by a term significance measure such as DF/ITF [Sparck Jones 1972]).
2. This vector is inserted into the `DMI Events` part of the Usage Model using an LDAP add operation.
3. The add event is handed over to the Scheduler, which is tightly integrated with the Directory Server. The Scheduler scans its internal subscription tables for matching entries. It finds a subscription of the ULC whose event type and basefilter match the current event.
4. The Scheduler asynchronously communicates the add event and associated data (mainly the event vector) to the ULC via CORBA, thereby following processing specifications that are also part of the subscription. Subsequently, the Scheduler resumes processing with step 6.
5. The ULC periodically checks its event queue and performs a univariate significance analysis with each of the received terms and finds, say, that the user is not interested in `Umweltbelastung`, with a normalized probability of 0.8, an individual probability of 0.04, and the classification 'yes'. If these values strongly differ from the ones in the current user's model, the ULC initiates an update via LDAP.
6. The Scheduler reports the successful submission to the Directory Server via interprocess communication, thereby completing the event submission.
7. The Client receives the result of its submission to the user modeling server via LDAP.

5 Evaluation

The development of a prototype of our LDAP-based User Modeling Server in the Deep Map project was a proof of concept for the feasibility of a directory-based approach. However, before user modeling servers can be deployed in real-world application scenarios with potentially millions of users, their runtime behavior must be experimentally tested under realistic workload conditions to ascertain their satisfactory performance in the target environment.

The parameters of such experiments, and specifically the simulated user interactions that cause requests to the UM server, should thereby closely resemble the interaction behavior at the target site in question. Unfortunately, most existing web traffic data are not very useful for our purposes. Many of them are based on proxy logs (e.g., [Duska et al. 1997; Gribble and Brewer 1997]) or web server logs (e.g., [Almeida et al. 1996; Padmanabhan and Qiu 2000]). While such data is an excellent basis for analyzing caching and pre-fetching strategies, it does not reflect all communication that would ordinarily take place between browsers and web servers [Fenstermacher and Ginsburg 2002]. For instance, browsers may connect to web servers via several proxies, and numerous caches may affect the amount of traffic between browsers and web servers. Most published studies are moreover based on websites of research institutions, which are not very representative for users' typical website visits[9] and presumably also not for the navigation behavior that is exhibited at more typical sites (see e.g. [Almeida et al. 1996; Padmanabhan and Qiu 2000]). Virtually all existing performance studies of UM servers also employed synthetic workloads rather than empirical web usage behavior (e.g., [VanderMeer et al. 2000; Datta et al. 2001]). The same holds true for performance studies of directory servers (e.g., [Keung and Abbott 1998; Wang et al. 2000]).

To avoid these limitations, we used findings from client-side studies of Internet usage behavior. We believe that these findings constitute a more promising basis for our model of real-world workload than the ones mentioned before. They provide an authentic view of users' online behavior, as opposed to the keyhole perspective of earlier proxy and server based studies.

5.1 Web Usage Patterns

[Rozanski et al. 2001] conducted a comprehensive analysis of click-stream data collected by the audience measurement service Nielsen//NetRatings. The data was collected at the *client* side from a panel of 2,466 Internet users over several months. First, the researchers identified 186,797 user sessions (defined as the time from when a user signs on to the Internet until she signs off, or ceases activity for more than an hour). Subsequently, they tested a variety of session characteristics with regard to their suitability for clustering these sessions. The most differentiating session characteristics were the following ones:

Session length: defined as the length of a single user session on the Internet.

Time per page: denotes the time interval between two subsequent web page requests.

Category concentration: the percentage of time a user stays at websites of the same category (e.g., news, sports, entertainment, real estate).

[9] For instance, [Nvision 1999] found that 35% of users' surfing time is spent at merely 50 (commercial) sites.

Site familiarity: the percentage of time a user stays at familiar sites, i.e. sites she had previously visited four or more times.

Based on these characteristics, Rozanski et al. carried out a cluster analysis and distinguished the following patterns of web usage (in parentheses their relative frequencies):

Quickie sessions (8%): These are short (one minute) visits to one or two 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 (15%): Here users seek and evaluate 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 (7%): Users focus on gathering specific information or completing concrete tasks (e.g., finding the website of a scientific conference and registering for it). They visit two websites 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, with 3.3 pages per site being visited.

Do It Again sessions (14%): These are focused on sites with which the user is familiar (e.g., online banks, chat rooms). Users spend about two minutes for each page. The average session lasts 14 minutes, with 2.1 sites and 3.3 pages per site being visited.

Loitering sessions (16%): Users visit familiar 'sticky' sites, such as news, gaming, telecommunications/ISP, and entertainment. Sessions last 33 minutes, with 8.5 sites and 1.9 pages per site being visited (two minutes per page on average).

Information Please sessions (17%): Users gather broad information from a range of often unfamiliar websites from several categories (e.g., they collect facts about a specific car model, find a dealership, negotiate a trade-in, and arrange a loan). Users visit 19.7 websites and 1.9 pages per site. The average session length is 37 minutes, and pages are viewed for one minute on average.

Surfing sessions (23%): They appear random, with users visiting nearly 45 sites in 70 minutes on average (about one 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 tasks 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.

5.2 Simulated workload

To test the performance of our UM server under different workload conditions, we simulated users' interaction with a hypothetical personalized website. Each user thereby follows one of the abovementioned session types. The content of each web page is characterized by 1-3 terms taken from the domain taxonomy (see Section 4.2.1.3). Web page requests by a user lead to add and query operations in her user profile on the UM server: the terms of the requested web page are processed and added to her interest model, and the user model is queried for terms that represent such interests, to personalize a requested webpage. As a shortcut though, we omit the web server in our simulation and represent web pages by their characteristic terms only.

To simulate different workload conditions, we systematically varied the following parameters:

- N (number of existing profiles in the UM server).
- W (number of web page requests per second).

For every factor combination, we generate a test plan with N user profiles. The behavior of currently active users of the hypothetical website is simulated by clients of our user modeling server. Clients are divided into seven classes, which represent the aforementioned session types. A class *i* comprises c_i clients which exhibit the web page request behavior that is characteristic for their class. The c_i clients of a class *i* create a total workload of w_i page requests per second. The combined workload of all clients equals the preset frequency of page requests W. We assume that w_i / W approximates the observed type frequency of class *i* (this assumption is corroborated by a manual count of the frequencies of Quickie and Just the Facts sessions at several German websites, such as the one described in [Fink et al. 2002]). Table 1 shows the test plan for a workload W of approximately 2 pages per second. Columns 2 and 3 contain the type frequency and the page request interval of the seven client classes from the study of [Rozanski et al. 2001]. Column 4 breaks down the workload W of two pages per second for each session type. For Quickies, for example, we calculate the number of clients c_i as 2 page requests per second * 15 seconds per page * 8% Quickies = 2 clients. Based on this, column 5 shows the actual workload w_i of the c_i clients for each session type. For Quickies, we calculate the workload w_i as 2 clients / 15 seconds per page = 0.13 pages per second.

Table 1: Simulation environment for W=2 page requests per second (* = figure rounded)

	Session type characteristics		Test bed parameters	
Variables / Session types	Relative type frequency	Interval between requests	No. of clients (c_i)*	Requests/sec. (w_i)*
Quickies	8%	15 sec	2	0.13
Just the Facts	15%	30 sec	9	0.30
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
Total	100%		146	2.00

We assigned a frequency property to each term in the domain taxonomy (see Section 4.2.1.3) that indicates how often it will occur as a characteristic term of simulated web pages. We assume that these frequencies are Zipf distributed, based on the fact that term frequency distributions in documents tend to follow Zipf's law [Zipf 1949]. We assigned another frequency property to simulated users that indicates how frequently they will start a new session with our hypothetical website. We assume that these frequencies are also Zipf distributed, based on several studies regarding the frequency and duration of people's Internet usage (e.g., [Patrick and Black 1996]). Finally, we also assume a Zipf distribution of the frequency in which web pages are requested by our simulated users, based on the observation that web page popularity follows 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. [Glassman 1994; Nielsen 1997; Breslau et al. 1999;

Padmanabhan and Qiu 2000] recommend different values for α. We followed [Padmanabhan and Qiu 2000] who analyzed the MSNBC news site since their study was the most recent and their site the most similar to our own target site. The authors recommend an α between 1.4 and 1.6, and hence we opted for α=1.5 and use this value for all three distributions.

Finally, we assume further that our UM server has to process the following operations for personalizing a requested web page[10]:

- *Three search operations* with Zipf-distributed terms from the domain taxonomy, namely for personalizing the page header (e.g., user-tailored banner ads), the navigation section (e.g., personalized links), and the content part (e.g., personalized news). We assume one exact search (such as for `cn=Natur`) and two substring searches (such as for `cn=Umwelt*`, which yields all those concepts that match this search filter (e.g., `Umwelt`, `Umweltbelastung`, `Umwelt-Ticket`)).
- *One add operation* for communicating the 1-3 characteristic terms of a web page as an interest event to the UM server.

5.3 Test Bed

Figure 10 shows our test bed. On the right side, we see the User Modeling System for Deep Map. Its representation part maintains the models that were described in Section 4.2.1, namely the User Model, Usage Model, System Model, and Service Model. We retained the three latter models from the Deep Map project without modification (including the taxonomy described in Section 4.2.1.3), but varied systematically the size of the User Model by setting the number of user profiles to 100, 500, 2,500, and 12,500. On the left side of Figure 10, we see the components that constitute our Test Bed, namely the Controller, Generators, Master, and Clients. In the following, we briefly describe their main tasks.

Controller. Its main tasks are

1. to create 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 user modeling system),
2. to execute test cases within the given constraints (e.g., test the runtime and ratio of different types of LDAP operations), and
3. to collect and record client-side measures (e.g., mean response times for LDAP add operations, and the average number of entries affected by LDAP search operations).

[10] See [Fink 2004] for additional details. Note that many personalized websites do not provide personalization on all pages, which reduces the load of the UM server.

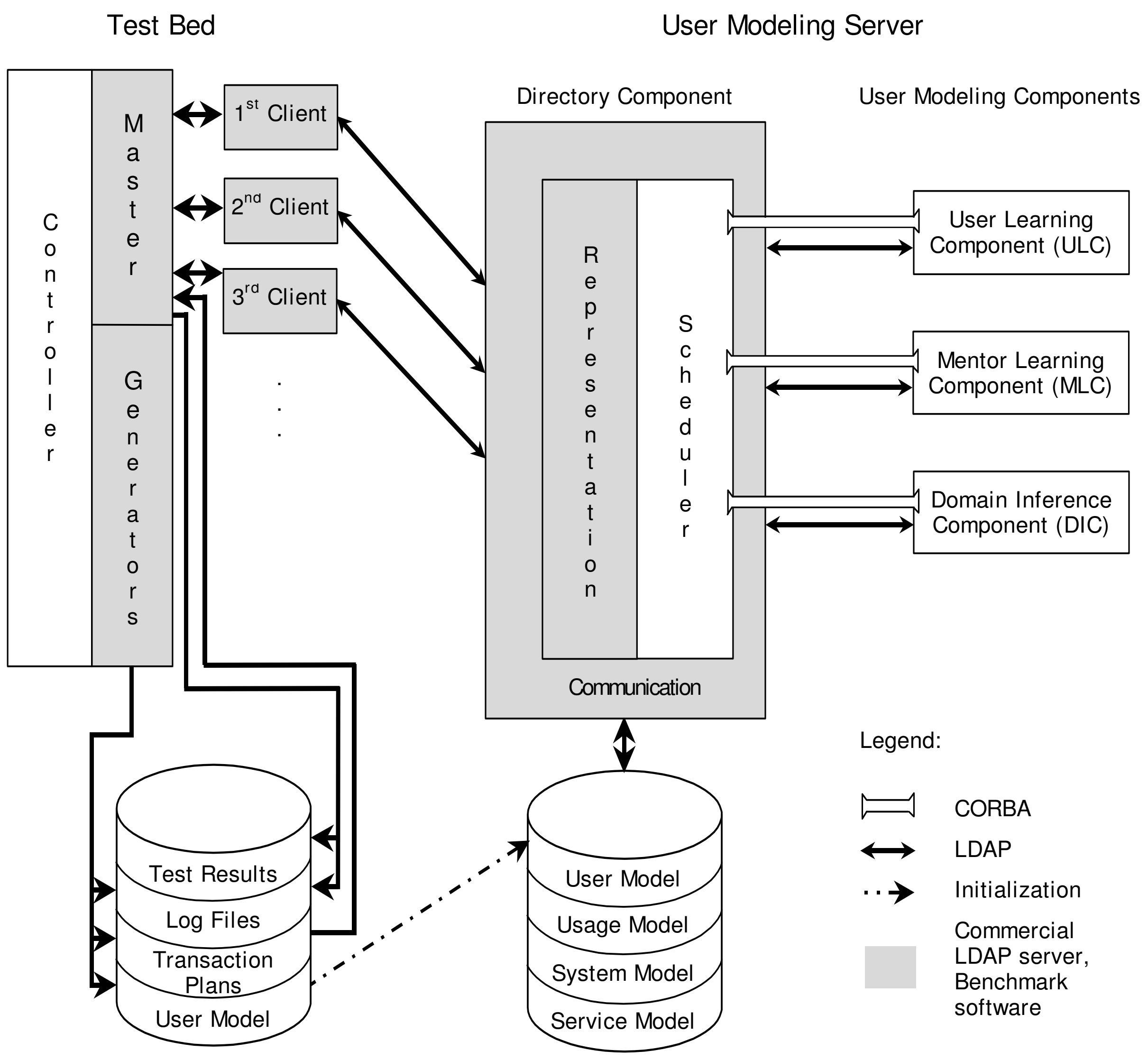


Figure 10: Overview of the experimental testbed

Generators. They create

1. user model contents (i.e., a preset number of user profiles using standard LDAP object types) ,
2. transaction plans, which specify the mix of LDAP operations to be sent to the UMS, and
3. 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 postal/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 users from the set of generated user profiles, and interests from the Deep Map Taxonomy, is controlled by our Zipf distribution.

Master. Its main tasks are

1. to start and initialize a preset number of Clients, each with a dedicated transaction plan,
2. to manage Clients at the time of testing, and
3. to compile Clients' individual performance measures into a single uniform report.

Clients. Their main tasks are

1. to execute their transaction plans (thereby submitting and monitoring LDAP requests), and
2. to report their performance measures back to the Master.

For implementing Generators, Master, and Clients, we took advantage of DirctoryMark [MindCraft 2006], a benchmark suite for LDAP servers. DirectoryMark simulates clients that simultaneously access an LDAP server and reports 269 performance indicators, all of which are measured from a client's point of view. Therefore, they do not only indicate the performance of the user modeling system 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 our user modeling extensions to standard LDAP object types (e.g., regarding interests and preferences), and the necessity for submitting interface events to the UMS. These modifications were realized by a wrapper around Directory Mark and 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.

For each test scenario, we generated an appropriate number of user profiles as well as transaction scripts that implement the workloads for each of the session types introduced earlier. For instance, a transaction plan for a Quickie client would look as follows:

1. log in (i.e., LDAP bind) to the UMS[11],
2. simulate a Web page request (i.e., submit three LDAP search operations and one LDAP add operation as described earlier),
3. wait for 15 seconds,
4. simulate another Web page request,
5. wait again for 15 seconds, and finally
6. log off from the UMS[11].

[11] We thereby assumed that Quickie applications (e.g., retrieval of stock quotes or sports results) will handle user logins/logoffs and authentication automatically, using cookies, certificates, and IP/Agent-related identification methods. The rejection rate for first-party cookies is currently at 1-4% [Webtrends 2005], hence an identification rate of more than 99% can be reached in combination with IP based methods.

During the execution of the experiment, the following steps were carried out under the supervision of the Master:

1. generate a User Model with a given 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,
3. reboot the servers, and
4. run each test case for 300 minutes.

The warm-up phase was introduced to avoid commencing a test run with all user profiles being empty, which might have unduly altered the average performance figures. The duration of the warm-up phase was determined in a pretest by observing the insert and update ratios in the User Model. We found that for 100 profiles these ratios converge to a stable base state after approximately 10 minutes, and we linearly increased this duration for higher numbers of profiles as follows: 50 minutes for 500 profiles, 250 minutes for 2,500 profiles, etc.

5.4 Small to Medium Scale Application Scenario

Our first series of experiments was carried out with a hardware configuration that would be typical for small web stores or news sites. In one test variant, all user modeling functionality resided on a single platform. In a second variant, we distributed the four components of our UM server across a network of four computers. In both conditions, a PC with an 800 MHz CPU, 512 MB of RAM and a 100 Mbps network card hosted the environment that simulated users submitting page requests. We varied our test parameters as follows:

- N (number of existing profiles in the UM server): 100, 500, 2,500, or 12,500[12].
- W (number of web page requests per second): 0.5, 1, 2, or 4[13].

5.4.1 Single Platform Tests

In the single platform tests, the complete UM server (i.e., Directory Component, ULC, MLC, and DIC) was running on a single PC with two 800 MHz processors, 1 GB of RAM, a RAID controller with two 18.3 GB UW-SCSI hard disks, and a 100 Mbps network card. The software used was Windows NT 4.0, iPlanet Directory Server 4.13 and VisiBroker 3.4. The learning and inference components were compiled with Java 1.2.2 and used the Java Hot Spot Server Virtual Machine 2.0.

[12] The corresponding user population is larger since not all users opt for personalization (5% in Yahoo and in a large German news portal [personal communication], and 64% in Excite [Excite 2006]).

[13] Based on data from [IVW 2006], one can estimate that two of three German websites with third-party traffic verification receive less than four page requests per second on average, even when only twelve usage hours per day and personalization on all pages are assumed.

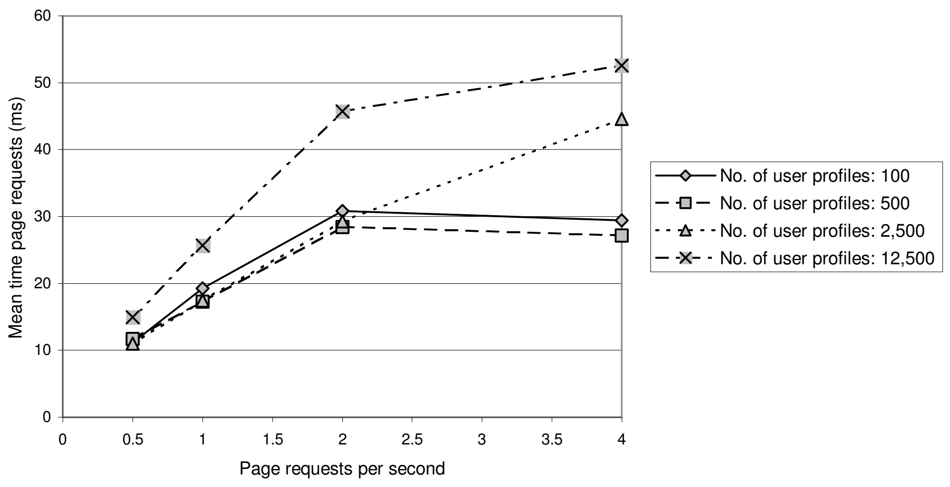


Figure 11: Mean processing times for personalizing a web page

Figure 11 shows the mean times that our UM server takes to perform the four user model operations for personalizing a page from the viewpoint of our hypothetical web application. The results for all 16 value combinations of our independent variables are charted. In general, mean times are only degressively proportional to the number of page requests and user profiles. In two cases (namely for 100 and 500 profiles), the response times for four page requests per second are even lower than for two. This advantageous behavior is mainly due to database caching in the LDAP directory server. The more user model operations are being sent to the server for a given number of user profiles, the faster this cache gets filled and the more operations can therefore be directly served from cache memory. We also see that all mean times for 12,500 users are higher than those for smaller numbers of user profiles, while the mean times for 100, 500, and 2,500 user profiles appear quite similar (except for 2,500 users and four pages). We assume that this effect results mainly from a higher hit rate (i.e., probability that a specific piece of information is contained in cache memory) in those cases that have a smaller number of user profiles. The overall performance and scalability of our UM server appears highly satisfactory. Even in the case of four page requests per second and 12,500 user models, the mean time to execute four user model operations and to return the results to 288 clients in parallel is smaller than 53 ms. The 99% confidence interval for the means does not exceed ± 0.24 ms due to the large sample size. The mean times plus one / two standard deviations never exceed 78 / 103 ms. A more detailed analysis shows that this graceful performance degradation occurs for both add and search operations. Since the overhead caused by the UM server is minor, web-based applications will be able to provide personalized services while responding within the desirable limit of one second and, in any case, the mandatory limit of ten seconds [Nielsen 1993]. The moderate surge of the mean response time when the number of clients and user profiles increases does not suggest impending performance cliffs and scalability limits.

5.4.2 Multiple Platform Tests

In the multi-platform scenario, only the Directory Component was running on the mentioned dual processor computer. The three other components of the UM server were each installed on a separate 600-800 MHz single processor PC with 100 Mbps network card. Figure 12 compares several measurements for both scenarios. 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, and its standard deviation to 10.54 from 24.92 milliseconds (i.e., nearly 60% in both cases). The single most important reason for this improvement is the considerably better search performance. The mean search time falls to 5.29 from 14.57 ms (-64%), and its σ to 5 from 13.57 ms (-63%). Less impressive is the performance gain of add operations: the mean time drops to 6.57 from 8.86 ms (-26%), and σ to 6 from 8.29 ms (-28%).

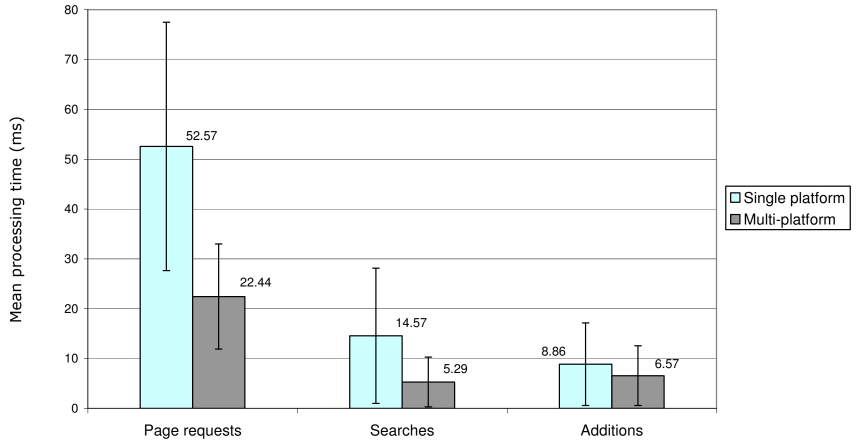


Figure 12: Single-platform vs. multi-platform performance
(12,500 profiles, 4 pages/sec)

The distribution of our UM server across a network of four computers improved its performance considerably. Search operations benefit most from the relieved dual processor computer, since they can now be carried out concurrently by the directory server. Add operations with their inherent need for multi-user synchronization [Fink 2004] can take less advantage of the additional hardware resources.

5.4.3 Evaluation of the Learning Components

So far, we discussed the performance of our UM server from the viewpoint of our hypothetical web application. Now we turn to the individual components of our server: the statistics-based User Learning Component, the similarity-based Mentor Learning Component, and the rule-based Domain Inference Component. These components operate concurrently to the Directory Component. Figure 13 shows the mean processing times of the ULC and the MLC for the single platform scenario. The performance of the DIC (which is comparable to that of the ULC) is discussed in [Fink and Kobsa 2002; Fink 2004].

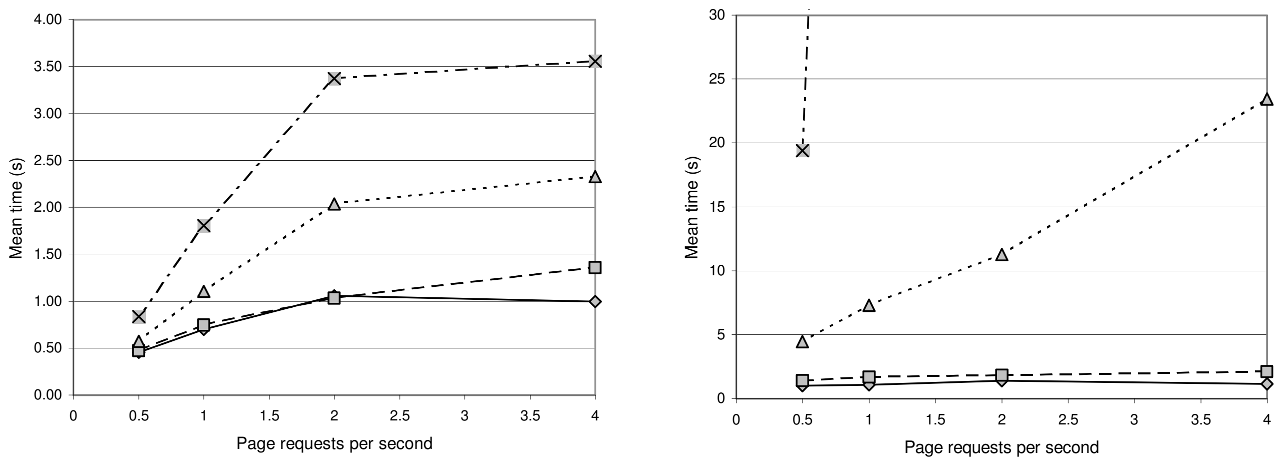


Figure 13: Mean processing times of statistics and similarity-based learning components (see Figure 11 for legend)

For the ULC, mean times seem to mainly depend on the number of user profiles. They grow degressively with increasing page requests, which is mainly due to the queue-based architecture of the ULC (it allows for bulk processing of submitted events and for interim storage of interest probabilities in the main memory, thereby saving costly updates of the user profile). All recorded mean times are smaller than four seconds, which is highly satisfactory since it permits keeping track of users' changing interests even between consecutive page requests. The ULC fully supports this inter-request learning for all session types and workloads we tested.

The performance of the MLC is less good. For 100, 500 and 2,500 profiles, all means are below 24 seconds but they grow progressively with increased page request rate. Except for Quickies, this still allows for a prediction of user interests and preferences between consecutive page requests. The response time deteriorates considerably though for 12,500 user profiles: 19 sec. for 0.5 and 141 sec. for 1 page/sec, but more than 2 hours for 2 and 4 pages/sec. In the latter two cases, the MLC presumably cannot keep pace with the stream of user arrivals and approaches its performance limits.

The most important reason for this weak scalability is the fact that the MLC we used in our tests searches for similar users in the whole user population. This results in low performance, large memory requirements, and often causes the underlying algorithm to be oversensitive to noise [Wilson and Martinez 2000]. In order to cope with these problems, more recent commercial versions of the UMS employ statistical sampling and reduction techniques from the area of instance-based learning like IB3, IB4 [Aha 1992], and DROP3 [Wilson and Martinez 1997, 2000]. Especially DROP3 often significantly narrows the search space to a reasonably sized sample of user profiles and can at the same time achieve higher prediction accuracies (see, e.g., [Wilson and Martinez 2000] for related experiments). Against this background, if we re-interpret the 100, 500, and 2,500 profiles used in our tests as qualified samples from a much larger set of user profiles, then the performance and scalability of the MLC seems again quite satisfactory.

Future work on the MLC may investigate the application of nearest-hyperrectangle, clustering, and partitioning algorithms to the matrix of users. For work on these topics, we refer to [Wettschereck 1994; Wettschereck and Dietterich 1995; Herlocker et al. 1999; O'Connor and Herlocker 1999].

5.5 Large Scale Application Scenario

The successful simulation results for a small to medium sized user-adaptive website put us in the position to run a series of experiments on a much larger scale. The most notable one comprised eight million user profiles[14] and a workload of approximately 42 web page requests per second[15]. To realize this workload, we employed a total of 1,794 simultaneous clients in several testbeds. The UM server was installed on a Fire V880 from Sun's entry-level server segment under Solaris 8, with eight 750 MHz processors, 8 MB cache per processor, 32 GB of RAM, and more than 200 GB of disk space. To take full advantage of the available hardware, we increased the cache size of the Directory Component and each learning component to 2 GB. The user modeling server was implemented in version 5.1 of iPlanet Directory Server. Otherwise the design of this experiment was very comparable to the one described in Section 3.1.

The results were again very encouraging. Our UM server showed a mean response time of 35 ms for personalizing a web page (i.e., for performing three LDAP searches and one add operation). This user modeling performance should easily allow a personalized application to stay well below the desirable response time limit of one second and, in any case, below the mandatory limit of ten seconds [Nielsen 2000]. None of the several million search and add operations that were submitted by our simulated users failed or timed out. Overall, the quality of service offered by our server seems highly satisfactory.

Our simulation environment obviously allows us to change any parameter of the experiment and to study the resulting effects. By systematic variation of the user modeling components we found, for instance, that their resource needs depend on their number (each can be present or absent, and instantiated multiple times), and 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 learning and inference components, seems to be a good solution. We also confirmed results from the literature regarding the effects of hardware sizing. For example, [Nelson 2002] mentions the following rules of thumb for the number of CPUs necessary to process 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".

5.6 Related work

At the beginning of Section 5, we discussed several performance studies whose traffic data are unlikely to accurately reflect the workloads of real-world user modeling servers, for various reasons (e.g., since synthetic workloads were used). The only study that seems comparable to

[14] As a comparison, AOL had about 20.1 million subscribers in the U.S. at the end of September 2005 [Goldman 2005]. The number of subscribed users is a more meaningful measure of comparison than the number of unique users, mainly due to the high mortality among the latter.

[15] Based on data from [IVW 2006], one can estimate that nine of ten German websites with third-party traffic verification receive less than 42 page requests per second on average, even when only twelve usage hours per day and personalization on all pages are assumed.

ours from a design point of view is the one described in [Carmichael et al. 2005]. The authors present a performance study of their PersonisLite user modeling server in a ubiquitous computing scenario. PersonisLite stores user models in the Berkeley DB database system. The server is meant to be used for a variety of applications, such as for generating recommendations in a museum [Bright et al. 2005], for location and activity modeling in a ubiquitous computing context [Carmichael et al. 2005; Whitaker and Kay 2005], and for author modeling in a computer-assisted tutoring context [Goldrei et al. 2005]

Like in our own study, the authors use recorded traffic data for simulations aimed at measuring the performance of their server, namely user login data from a campus environment and periodic scan data whether they are still active. The experiment resulted in small CPU times for add operations and linear increase with the number of items added, and also small CPU times for two types of search operations with different complexities, with virtually no increase with the number of items in the user model. Unfortunately these results are not comparable to ours, e.g. for the following reasons:

1. Their server hardware is quite different from ours (a single Intel Pentium IV processor with 2,53 GHz versus, e.g., a dual Xeon processor configuration running at 800 MHz in our case).
2. The number and complexity of PersonisLite services seems to be smaller than those on our UM server. Specifically, the PersonisLite study focuses on a comparison of the performance of two different inference processes ('resolvers'), namely ones that only take the latest few user data into account with ones that process the last half hour's worth of data. The UMS study evaluates instead the user modeling components introduced in Section 4.1.
3. Their test approach lacks a two-factor design, with an increasing number of data items and number of client applications (see Section 5.4).
4. Their test starts with an empty database, whereas our database is pre-filled in a dedicated warm-up phase so that the server performance does not become favorably biased at the beginning of the experiment due to a smaller than normal database size (see Section 5.3).
5. They successively submit large batches of user model operations of the same type (e.g., 450,000 add operations, 400 ask operations) rather than continuously mixing them (see Section 5.2), which may unduly boost the performance due to caching effects.
6. They only collect a few measures for server performance (e.g., CPU time) directly on the user modeling server, whereas we collect 269 different performance measures both on the user modeling server and on the client side (see Section 5.3).
7. Their central measure for evaluating server performance is CPU time (i.e. the consumption of a single resource on the server), whereas the central measure in our test is the response time for user modeling operations from a client point of view (i.e., including server performance, network latency, and client performance).

6 Summary and Conclusion

We showed that the use of directories for storing user information offers significant advantages over the two traditional approaches in academia and industry, namely flat file systems and database systems. These advantages lie particularly in the

- management and retrieval of (user-related) information, in a way that is compliant with established standards;
- pre-defined user related information types, and the possibility to define new types;
- distribution of information across a network, which often leads to better performance, scalability, availability, and reliability of the user modeling service;
- replication and loose synchronization 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;
- ability to realize a virtually centralized distributed architecture for a user model repository; and the
- security of information and users' privacy, by providing facilities for authentication, signing, encryption, access control, auditing, and resource control (see [Fink 2004]).

We presented the architecture of our user modeling server UMS which takes advantage of the above benefits of LDAP. We also briefly demonstrated the utility of this server in an application scenario. In simulation experiments we 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 quite moderate. These results complement and corroborate the discussion of Section 3 regarding the advantages of directory systems as a basis for user modeling servers.

Since the workload used in the simulations was based on empirically gathered usage data and statistical findings from web usage research and information retrieval, the ecological relevance of our experiments appears to be high. Virtually all parameters of our experiment (e.g., the number of user modeling components used, their computational characteristics, their distribution across platform, and the characteristics of the site platform, the user behavior and the webpage content) can be systematically changed and the resulting effects studied. Our experience with actual installations of our server in commercial environments showed that this approach and the developed simulation test bed were an indispensable tool for real-world personalization.

More recent experience that we gained from deploying our user modeling server to large commercial Web sites confirms that our server can indeed be deployed in high-workload environments as well. Our user modeling server has been already successfully deployed in commercial application environments. A profiling application across most German Top 100 web sites with a total workload of tens of millions of users and several billion page impressions per month is already up and running on a farm of high-end Xeon-based servers. Pilot systems for other European countries are currently being tested.

7 Acknowledgment

This work has been supported by the European Media Laboratory (EML), the National Science Foundation (Grant DST 0307504), and by an Alexander von Humboldt Research Award. We would like to thank the anonymous reviewers for their valuable comments on an earlier version of this paper.

8 References

Aha, D. W. (1992). "Tolerating Noisy, Irrelevant and Novel Attributes in Instance-Based Learning Algorithms." *International Journal of Man-Machine Studies* 36: 267-287. DOI: 10.1016/0020-7373(92)90018-G.

Almeida, V., A. Bestavros, M. Crovella and A. Oliveira (1996). Characterizing Reference Locality in the WWW. Fourth International Conference on Parallel and Distributed Information Systems, IEEE Computer Society, 92-103. DOI: 10.1109/PDIS.1996.568672.

Andrews, G. (1991). "Paradigms for Process Interaction in Distributed Programs." *ACM Computing Surveys* 23(1): 49-90. DOI: 10.1145/103162.103164.

Bigfoot (2006). Bigfoot. http://search.bigfoot.com/.

Borland (2006). "Borland VisiBroker." http://www.borland.com/us/products/visibroker/.

Bosch, G. (1988). ASCON. Memo, SFB 314: AI – Knowledge-Based Systems, Dept. of Computer Science, Saarland University, Saarbrücken, Germany.

Brajnik, G. and C. Tasso (1994). "A Shell for Developing Non-monotonic User Modeling Systems." *International Journal of Human-Computer Studies* 40: 31-62. DOI: 10.1006/ijhc.1994.1003.

Breese, J., D. Heckerman and C. Kadie (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering. Proc. of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98), San Francisco, Morgan Kaufmann, 43-52. ftp://ftp.research.microsoft.com/pub/tr/tr-98-12.pdf.

Breslau, L., P. Cao, L. Fan, G. Phillips and S. Shenker (1999). Web Caching and Zipf-Like Distributions: Evidence and Implications. INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, 126-134. DOI: 10.1109/INFCOM.1999.749260.

Bright, A., J. Kay, D. Ler, K. Ngo, W. Niu and A. Nuguid (2005). Adaptively Recommending Museum Tours. UBICOMP-05 Workshop on Smart Environments and their Applications to Cultural Heritage, Tokyo, Japan. http://smart.arces.unibo.it/pdf/04-Adaptively-Recommending_Bright.pdf.

Brusilovsky, P., S. Ritter and E. Schwarz (1997). Distributed Intelligent Tutoring on the Web. AI-ED'97, 8th World Conference on Artificial Intelligence in Education, Kobe, Japan, 482-489. http://www2.sis.pitt.edu/~peterb/papers/AIED97.html.

Carmichael, D. J., J. Kay and B. Kummerfeld (2005). "Consistent Modelling of Users, Devices and Sensors in a Ubiquitous Computing Environment." *User Modeling and User-Adapted Interaction: The Journal of Personalization Research* 15(3-4): 197-234. DOI: 10.1007/s11257-005-0001-z.

Chadwick, D. (1996). Understanding X.500: The Directory. London, Thomson.

Critical Path (2006). Critical Path. http://www.cp.net.

Datta, A., K. Dutta, D. VanderMeer, K. Ramamritham and S. B. Navathe (2001). "An Architecture to Support Scalable Online Personalization on the Web." *The VLDB Journal* 10: 104–117. DOI: 10.1007/s007780100037.

Deep Map (2001). Deep Map: Intelligent, Mobile, Multi-Media and Full of Knowledge (Project Homepage). http://www.eml.org/english/research/deepmap/deepmap.html.

Duska, B. M., D. Marwood and M. J. Feeley (1997). The Measured Access Characteristics of World-Wide-Web Client Proxy Caches. USENIX Symposium on Internet Technologies and Systems, Monterey, CA. http://www.usenix.org/publications/library/proceedings/usits97/duska.html.

enQuire (2006). enQuire Identity Server. http://www.persistentsys.com/products/enquire/enquire.htm.

Excite (2006). Excite Network Online Media Kit. http://www.excitenetwork.com/advertising/index/id/Directmarket|ListRental|3|1.html.

Fenstermacher, K. D. and M. Ginsburg (2002). Mining Client-Side Activity for Personalization. Fourth Workshop on Advanced Issues in Electronic Commerce and Web Information Systems (WECWIS), Newport Beach, CA, 44-51. http://linux.ece.uci.edu/TFEC/wecwis.html.

Finin, T. W. (1989). GUMS: A General User Modeling Shell. In: A. Kobsa and W. Wahlster, eds: User Models in Dialog Systems. Berlin, Heidelberg, Springer-Verlag**:** 411-430.

Finin, T. W. and D. Drager (1986). GUMS1: A General User Modeling System. Sixth Canadian Conference on Artificial Intelligence, Montreal, Canada, 24-29.

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, Springer-Verlag**:** 191-200. http://bistrica.usask.ca/UM/UM99/Proc/fink.pdf.

Fink, J. (2004). User Modeling Servers - Requirements, Design, and Evaluation. Amsterdam, Netherlands, IOS Press. http://books.google.com/books?q=isbn:1586034057.

Fink, J. and A. Kobsa (2000). "A Review and Analysis of Commercial User Modeling Servers for Personalization on the World Wide Web." *User Modeling and User-Adapted Interaction: The Journal of Personalization Research* 10(2-3): 209-249. DOI: 10.1023/A:1026597308943.

Fink, J. and A. Kobsa (2002). "User Modeling in Personalized City Tours." *Artificial Intelligence Review* 18(1): 33-74. DOI: 10.1023/A:1016383418977.

Fink, J., J. Koenemann, S. Noller and I. Schwab (2002). "Putting Personalization into Practice." *Communications of the ACM* 45(5): 41-42. DOI: 10.1145/506218.506242.

Glassman, S. (1994). "A Caching Relay for the World Wide Web." *Computer Networks and ISDN Systems* 27(2): 165-172. DOI: 10.1016/0169-7552(94)90130-9.

Goldman, A. (2005). "Top U.S. ISPs by Subscriber: How We Count." *ISP-Planet*. http://www.isp-planet.com/research/rankings/2005/usa_insight_q32005.html.

Goldrei, S., J. Kay and B. Kummerfeld (2005). Exploiting User Models to Automate the Harvesting of Metadata for Learning Objects. AIED-05 Workshop on Adaptive Systems for Web-Based Education: Tools and Reusability, Amsterdam, Netherlands. http://www.lcc.uma.es/~eva/waswbe05/papers/goldrei.pdf.

Goodman, B., F. Linton and J. Schoening (1999). Workshop on Standards for Learner Modeling. http://www.cs.usask.ca/UM99/w2.shtml.

Gribble, S. D. and E. A. Brewer (1997). System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. USENIX Symposium on Internet Technologies and Systems, Monterey, CA. http://www.usenix.org/publications/library/proceedings/usits97/gribble.html.

Heckmann, D., T. Schwartz, B. Brandherm, M. Schmitz and M. von Wilamowitz-Moellendorff (2005). GUMO: The General User Model Ontology. In: L. Ardissono, P. Brna and A. Mitrovic, eds: User Modeling 2005: 10th International Conference, UM 2005, Edinburgh, Scotland.**:** 428-432. DOI: 10.1007/11527886_58.

Herlocker, J., J. Konstan, A. Borchers and J. Riedl (1999). An Algorithmic Framework for Performing Collaborative Filtering. Proc. of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval., New York, 230-237. DOI: 10.1145/312624.312682.

Hill, P. and J. Lloyd (1993). The Gödel Programming Language. Cambridge, MA, MIT Press.

Howes, T., M. Smith and G. Good (1999). Understanding and Deploying LDAP Directory Services. Indianapolis, IN, Macmillan.

Howes, T. A. and M. Smith (1997). LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol. Indianapolis, IN, Macmillan.

IBM (2006a). IBM Lotus Notes. http://www.ibm.com/notes.

IBM (2006b). IBM Tivoli Directory Server. http://www-306.ibm.com/software/tivoli/products/directory-server/.

Informix (2006). Informix Product Family. http://www.ibm.com/software/data/informix/.

ISO (1989). Information technology -- Database languages -- SQL. ISO/IEC 9075:1989, International Standardization Organization, Geneva, Switzerland. http://www.iso.org.

ISO (2003). Information technology -- Database languages -- SQL. ISO/IEC 9075:2003, International Standardization Organization, Geneva, Switzerland. http://www.iso.org.

ITU-T (2001). Information Technology – Open Systems Interconnection – The Directory: Overview of Concepts, Models and Services. Recommendation X.500 (02/01), International Telecommunication Union. http://www.itu.int/ITU-T/publications/recs.html.

IVW (2006). IVW Online Usage Data March 2006 (in German). http://www.ivwonline.de/ausweisung2/suchen.php.

Kay, J. (1990). um: a Toolkit for User Modelling. Second International Workshop on User Modeling, Honolulu, HI.

Kay, J. (1995). "The um Toolkit for Reusable, Long Term User Models." *User Modeling and User-Adapted Interaction: The Journal of Personalization Research* 4(3): 149-196. DOI: 10.1007/BF01100243.

Kay, J., B. Kummerfeld and P. Lauder (2002). Personis: A Server for User Models. In: P. De Bra, P. Brusilovsky and R. Conejo, eds: Adaptive Hypermedia and Adaptive Web-Based Systems: Second International Conference, AH 2002. Berlin Heidelberg, Springer-Verlag**:** 203–212. http://springerlink.metapress.com/link.asp?id=2l54yrgc0p8n2d5g.

Keung, S. and S. Abbott (1998). LDAP Server Performance Report. http://www.bnelson.com/sizing/docl/ldapsPerformance.html.

Kobsa, A. (1990). "Modeling The User's Conceptual Knowledge in BGP-MS, a User Modeling Shell System." *Computational Intelligence* 6: 193-208.

Kobsa, A. (1991). Utilizing Knowledge: The Components of the SB-ONE Knowledge Representation Workbench. In: J. Sowa, ed. Principles of Semantic Networks: Exploration in the Representation of Knowledge. San Mateo, CA, Morgan Kaufmann**:** 457-486.

Kobsa, A. (2001). "Generic User Modeling Systems." *User Modeling and User-Adapted Interaction: The Journal of Personalization Research* 11(1-2): 49-63. DOI: 10.1023/A:1011187500863.

Kobsa, A., J. Koenemann and W. Pohl (2001). "Personalized Hypermedia Presentation Techniques for Improving Customer Relationships." *The Knowledge Engineering Review* 16(2): 111-155. DOI: 10.1017/S0269888901000108.

Kobsa, A., D. Müller and A. Nill (1994). KN-AHS: An Adaptive Hypertext Client of the User Modeling System BGP-MS. Proceedings of the Fourth International Conference on User Modeling, Hyannis, MA, 99-105. Reprinted in M. Maybury and W. Wahlster, eds. (1998). Readings in Intelligent User Interfaces. San Mateo, CA: Morgan Kaufman, 372-378. http://www.ics.uci.edu/~kobsa/papers/1994-UM94-kobsa.pdf.

Kobsa, A. and W. Pohl (1995). "The BGP-MS User Modeling System." *User Modeling and User-Adapted Interaction: The Journal of Personalization Research* 4(2): 59-106. DOI: 10.1007/BF01099428.

Kobsa, A., W. Pohl and J. Fink (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. and J. Schreck (2003). "Privacy through Pseudonymity in User-Adaptive Systems." *ACM Transactions on Internet Technology* 3(2): 149–183. DOI: 10.1145/767193.767196.

Kummerfeld, R. and J. Kay (1997). Remote Access Protocols for User Modelling. Proceedings and Resource Kit for Workshop User Models in the Real World, Chia Laguna, Sardinia, 12-15. http://www.cs.usyd.edu.au/~judy/Homec/Pubs/1997_umnet.html.

Liberty (2006). Liberty Alliance Project: Digital Identity Defined. http://www.projectliberty.org/.

Loshin, P. (2000). Big Book of Lightweight Directory Access Protocol (LDAP) RFCs. San Diego, CA, Morgan Kaufmann.

LTSC (2006). Learning Technology Standards Committee. http://ieeeltsc.org/.

Malaka, R. and A. Zipf (2000). DEEP MAP – Challenging IT Research in the Framework of a Tourist Information System. In: D. Fesenmaier, S. Klein and D. Buhalis, eds: Informaton and Communication Technologies in Tourism 2000: Proceedings of ENTER 2000. Wien, New York, Springer**:** 15-27.

McCune, W. W. (1994). OTTER 3.0 Reference Manual and Guide. In: Argonne National Laboratory, Mathematics and Computer Science Division. Argonne, IL. http://www-unix.mcs.anl.gov/AR/otter/.

Microsoft (2006a). Microsoft Exchange Server. http://www.microsoft.com/exchange/.

Microsoft (2006b). Windows Server 2003 Active Directory. http://www.microsoft.com/windowsserver2003/technologies/directory/activedirectory/default.mspx.

Miller, B. N., J. A. Konstan and J. Riedl (2004). "PocketLens: Toward a Personal Recommender System." *ACM Transactions on Information Systems* 22(3): 437-476. DOI: 10.1145/1010614.1010618.

MindCraft (2006). DirectoryMark: The LADP Server Benchmarking Tool. http://www.mindcraft.com/directorymark/.

Mitchell, T. (1997). Machine Learning. New York, NY, McGraw-Hill.

Nelson, B. (2002). Sizing Guide for Netscape Directory Server. http://www.bnelson.com/sizing/doc2/Directory4_0-SizingGuide.html.

Nielsen, J. (1993). Usability Engineering. San Diego, CA, Academic Press.

Nielsen, J. (1997). Zipf Curves and Website Popularity. http://www.useit.com/alertbox/zipf.html.

Nielsen, J. (2000). Designing Web Usability. Indianapolis, IN, New Riders.

Novell (2006). Novell eDirectory. http://www.novell.com/products/edirectory/.

Nvision (1999). 35 Percent of Surfing Time is Spent on 50 Sites. http://www.nua.com/surveys/index.cgi?f=VS&art_id=905355323&rel=true.

O'Connor, M. and J. Herlocker (1999). Clustering Items for Collaborative Filtering. Proceedings of the ACM SIGIR Workshop on Recommender Systems, Berkeley, CA. http://web.engr.oregonstate.edu/~herlock/papers/sigir99_workshop_clustering.pdf.

OMG (2001). Object Management Group (OMG). http://www.omg.org.

Orfali, R., D. Harkey and J. Edwards (1994). Essential Client/Server Survival Guide. New York, Wiley and Sons.

Orwant, J. (1993). Doppelgänger Goes to School: Machine Learning for User Modeling. Master Thesis, MIT, Cambridge, MA.

Orwant, J. (1994). Privacy and User Models: Threats, Caveats, and Safeguards. http://citeseer.ist.psu.edu/orwant94privacy.html.

Orwant, J. (1995). "Heterogenous Learning in the Doppelänger User Modeling System." *User Modeling and User-Adapted Interaction: The Journal of Personalization Research* 4(2): 107-130. DOI: 10.1007/BF01099429.

Padmanabhan, V. and L. Qiu (2000). The Content and Access Dynamics of a Busy Web Site: Findings and Implications. ACM SIGCOMM, ACM, 111-123. DOI: 10.1145/347059.347413.

Paiva, A. and J. Self (1994). TAGUS: A User and Learner Modeling System. In: Proc. of the Fourth International Conference on User Modeling. Hyannis, MA**:** 43-49.

Paiva, A. and J. Self (1995). "TAGUS -- A User and Learner Modeling Workbench." *User Modeling and User-Adapted Interaction: The Journal of Personalization Research* 4(3): 197-226. DOI: 10.1007/BF01100244.

PAPI (2001). PAPI Learner, Draft 8 Specification. http://edutool.com/papi.

Passport (2006). Microsoft Passport Network. http://www.passport.net.

Patrick, A. S. and A. Black (1996). "Implications of Access Methods and Frequency of Use for the National Capital Freenet." http://debra.dgbt.doc.ca/services-research/survey/connections/.

Pereira, F., Ed. (1996). C-Prolog User's Manual Version 1.5. http://www.cs.duke.edu/~raw/cps106/cprolog.ps.

Persistent (2006). Persistent. http://www.persistentsys.com.

Pohl, W. (1998). Logic-Based Representation and Reasoning for User Modeling Shell Systems. Sankt Augustin, Germany, infix.

Pohl, W., I. Schwab and I. Koychev (1999). Learning About the User: A General Approach and Its Application. IJCAI'99 Workshop Learning About Users, Stockholm, Sweden.

Pope, A. (1997). The Corba Reference Guide: Understanding the Common Object Request Broker Architecture. Sydney, Australia, Addison-Wesley.

Razmerita, L., A. Angehrn and A. Maedche (2003). Ontology-Based User Modeling for Knowledge Management Systems. In: P. Brusilovsky, A. Corbett and F. De Rosis, eds: User Modeling 2003: 9th International Conference, UM 2003. Heidelberg, Germany, Springer Verlag**:** 213-217. http://springerlink.metapress.com/link.asp?id=thw9rmvmvklx9hac.

Rozanski, H., G. Bollman and M. Lipman (2001). Seize the Occasion: Usage-based Segmentation for Internet Marketers. http://www.strategy-business.com/media/pdf/03-20-01_eInsight.pdf.

Schreck, J. (2003). Security and Privacy in User Modeling. Dordrecht, Netherlands, Kluwer Academic Publishers. http://www.security-and-privacy-in-user-modeling.info.

Schwab, I. and W. Pohl (1999). Learning Information Interest from Positive Examples. UM99 Workshop on Machine Learning for User Modeling, Banff, Canada.

Shukla, S. and A. Deshpande (2000). Tutorial: LDAP Directory Services – Just Another Database Application? 2000 ACM SIGMOD International Conference on Management of Data, New York, NY. http://www.pspl.co.in/presentation/sigmod2000_directory_database_tutorial.pdf.

Sparck Jones, K. (1972). "A Statistical Interpretation of Term Specificity and its Application to Retrieval." *Journal of Documentation* 28: 11-21. DOI: 10.1108/00220410410560573.

Sun (2006). Sun Java System Directory Server Enterprise Edition. http://www.sun.com/software/products/directory_srvr_ee/.

Switchboard (2006). Switchboard. http://www.switchboard.com/.

Tornago (2006). Net Perceptions. http://www.tornago.com.

VanderMeer, D., K. Dutta and A. Datta (2000). Enabling Scalable Online Personalization on the Web. 2nd ACM Conference on Electronic Commerce, Minneapolis, MN, ACM, 185-196. DOI: 10.1145/352871.352892.

Vassileva, J., G. McCalla and J. Greer (2003). "Multi-Agent Multi-User Modeling in I-Help." *User Modeling and User-Adapted Interaction: The Journal of Personalization Research* 13(1+2): 179-210. DOI: 10.1023/A:1024072706526.

Vergara, H. (1994). PROTUM: A Prolog Based Tool for User Modeling. WIS-Report 10, WG Knowledge-Based Information Systems, Department of Information Science, University of Konstanz, Germany.

Wahl, M., T. Howes and S. Kille (1977). Lightweight Directory Access Protocol (v3). RFC 2251, Internet Engineering Task Force. http://www.ietf.org/rfc/rfc2251.txt.

Wang, X., H. Schulzrinne, D. Kandlur and D. Verma (2000). Measurement and Analysis of LDAP Performance. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, ACM, 156-165. http://www.cs.columbia.edu/~xinwang/public/paper/ldap_sigmetrics.pdf.

Webtrends (2005). WebTrends Customers Switch to First-Party Cookies and See Accuracy Skyrocket by More Than 300 Percent. http://www.webtrends.com/AboutWebTrends/NewsRoom/NewsRoomArchive/2005/WebTrendsCustomersSwitchtoFirst-PartyCookiesandSeeAccuracySkyrocketbyMoreThan300Percent.aspx.

Weltman, R., C. Tomlinson and S. Sonntag (2005). The Java LDAP Application Program Interface. http://www.ietf.org/internet-drafts/draft-ietf-ldapext-ldap-java-api-19.txt.

Wettschereck, D. (1994). A Hybrid Nearest-Neighbor and Nearest-Hyperrectangle Algorithm. Proceedings of the 7th European Conference on Machine Learning, Catania, Italy, Springer-Verlag, 323-335.

Wettschereck, D. and T. G. Dietterich (1995). "An Experimental Comparison of Nearest-Neighbor and Nearest-Hyperrectangle Algorithms." *Machine Learning* 19(1): 5-28. DOI: 10.1007/BF00994658.

Whitaker, R. and J. Kay (2005). Location and Activity Modelling in Intelligent Environments. UM05 Workshop on Decentralized, Agent Based and Social Approaches to User Modelling, Edinburgh, Scotland. http://www.l3s.de/~dolog/dasum/Whitaker_Kay_um05.pdf.

WhitePages.com (2006). WhitePages.com. http://www.whitepages.com.

Wilson, D. R. and T. R. Martinez (1997). Instance Pruning Techniques. In: D. Fisher, ed. Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97). San Francisco, CA, Morgan Kaufmann Publishers: 403-411. http://synapse.cs.byu.edu/papers/wilson.icml97.prune.pdf.

Wilson, D. R. and T. R. Martinez (2000). "Reduction Techniques for Instance-Based Learning Algorithms." *Machine Learning* 38: 257-286. DOI: 10.1023/A:1007626913721.

Yaacovi, Y., M. Wahl and T. Genovese (1999). Lightweight Directory Access Protocol (v3): Extensions for Dynamic Directory Services. RFC 2589, Internet Engineering Task Force. http://www.ietf.org/rfc/rfc2589.txt.

Yimam, D. and A. Kobsa (2003). Expert Finding Systems for Organizations: Problem and Domain Analysis and the DEMOIR Approach. In: M. Ackerman, A. Cohen, V. Pipek and V. Wulf, eds: Beyond Knowledge Management: Sharing Expertise. Cambridge, MA, MIT Press. http://www.ics.uci.edu/~kobsa/papers/2003-JOCEC-kobsa.pdf.

Yodlee (2006). Yodlee. http://www.yodlee.com.

Young, A. (1995). Connection-Less Lightweight X.500 Directory Access Protocol. RFC 1798, Internet Engineering Task Force. http://www.ietf.org/rfc/rfc1798.txt.

Zipf, G. K. (1949). Human Behavior and the Principle of Least Effort. Reading, MA, Addison-Wesley.