GUMSAWS: A Generic User Modeling Server for Adaptive Web Systems

by

Jie Zhang

BCS, University of New Brunswick, 2003

A Thesis Submitted in Partial Fulfillment of

the Requirements for the Degree of

Master of Computer Science

in the Graduate Academic Unit of

Faculty of Computer Science

Supervisor: Ali A. Ghorbani, Ph.D., Computer Science

Examining Board: Dawn MacIsaac, Ph.D., Computer Science & Electrical and Computer Engineering, Chair
Stephen Marsh, Ph.D., Computer Science, UNB & National Research Council-IIT
Greg Fleet, Ph.D., Faculty of Business, UNB Saint John

This thesis is accepted by the

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

April 2005

Abstract

This thesis focuses on the architecture, design and implementation of GUMSAWS, a generic user modeling server for adaptive Web systems. GUMSAWS is proposed based on the user modeling tasks performed by the adaptive Web systems implemented from an adaptive Web site construction framework. This framework describes the interplay between the content of a Web site, the usage of that content, the users who consume the information, and the packaging and navigation structure of that content. To describe the information of an individual user or group of users, the adaptive Web framework requires the support of user modeling technology. In early work, user modeling tasks were performed without the aim of modifiability and reusability. These user modeling tasks are unable to fulfil the needs of the adaptive Web site construction framework.

GUMSAWS is designed and implemented to reach the goals of generality, extendability, and replaceability. GUMSAWS acts as a centralized user modeling server to assist several adaptive Web systems concurrently. It offers the user modeling functions of building up a user/group profile, and storing, retrieving, updating and deleting entries. It provides the user modeling tasks of inferring user property values and providing adaptive systems with recommendations according to users' interests. GUMSAWS also provides a facility to allow users to see and modify their profiles.

GUMSAWS is examined within the two application domains, E-tailer and E-news. The accuracy of inferring user property values from different information sources (direct, groups information, association rules and general facts) is evaluated in E-tailer domain. The example application in E-news domain, PENS (Personalized Electronic News System), also illustrates the basic user modeling functions provided by GUMSAWS, and its user modeling task of providing recommendations according to users' interests.

Contents

List of Tables

List of Figures

Acknowledgements

I would first like to acknowledge Dr. Ali A. Ghorbani, my supervisor, for the opportunity to work on his project, Adaptive Websites, and for his constant guidance, support and patience. Many thanks go to Mark Kilfoil for providing KDDCUP2000 data for the evaluation of the system. Thanks to my group members Mehran Najarbashi and Hossein Sadat for cooperation of implementation of PENS. I am also grateful for the technical discussions from Wenpu Xin and Zhong Lei.

Finally, I would like to extend my heartfelt thanks to my family in China: my mother Aihua, my father Guoping, my older sister Hui, and my younger brother Fan, for their lifelong love, encouragement and motivation in all that I do.

Chapter 1

Introduction

Web sites are becoming more and more popular and convenient for providing information over a broad number of topics. They are used in diverse systems, such as educational systems, on-line information systems, on-line help systems, information retrieval systems, and e-commerce systems. Problems occur as the use of Web sites increases, their size gets larger, and their structure becomes more complex. The rich link structure of a hypermedia application can cause users to get easily overwhelmed by the sheer number of navigation choices, and they may become unable to navigate effectively. This is referred to as the "lost-in-hyperspace" problem of navigating the Web. Web sites provide (from the point of view of a user) relatively static content, though they are viewed by diverse users. This may cause difficulties for those who have less background, may be redundant for those who are familiar with the information, and may present more uninteresting than interesting material for others. This is a "one-size-fits-all" problem common to non-adaptive Web sites.

To address these problems, there is a demand for Web sites to be automatically adapted to reach the goals of personalization, recommendation, selection, and usage analysis [23]. These Web sites are called adaptive Web sites. They are able to transform a page request into a final page response by considering information about the page requested, about the user, about the way the system has been used, about the

environment of the site, and about the environment of the user. They may provide three different types of adaptation: content adaptation, navigation adaptation, and presentation adaptation. While there is some overlap between these types of adaptation, they are different in what they target. Content adaptation adds and/or removes information fragments to/from the page based on the current context. Navigation (structure) adaptation adds, removes, hides, sorts links, and/or changes the colour of the links in a page, in order to provide the best navigation structure in the current context for the user. Adding a recommended item at the end of a page, also falls in this class of adaptation. Finally, presentation adaptation reformats the information fragments to achieve the appropriate final presentation for the current context.

We, the Intelligent and Adaptive Systems (IAS) group at the Faculty of Computer Science, University of New Brunswick, developed an adaptive Web site construction framework [14]. This framework provides an architecture based on which an adaptive Web system for different domains, can be built. It describes the interplay between the content of a Web site, the usage of that content, the users who consume the information, and the packaging and navigation structure of that content. The framework requires usage mining to extract usage information after the reconstruction of user sessions from server logs. The reconstruction of user sessions is performed based on our previous work using improved time-oriented heuristics [46], referrer-based heuristics [37], and a Markov chain model combined with a competitive algorithm [31]. The adaptive Web framework also requires the support of user modeling technology to describe the information of an individual user or group of users from the mined usage information. User modeling in this context should be able to offer the function of incrementally building up a user model, the function of storing, updating and deleting entries, and the function of maintaining the consistency of the model [29]. It may also contain the tasks of inferring user property values and providing recommendations based on users' interests.

1.1 Motivation

The Adaptive Web Site Construction framework requires user modeling technology. The framework has the goals of versatility, flexibility, extendability and scalability. As a result, user modeling should be modifiable, reusable, and general and applicable to as many application domains as possible.

User modeling has been performed by user modeling components, shell systems, and servers. Traditionally, user modeling components are embedded into application systems. Embedded user modeling components lack reusability and are only applicable to the adaptive system to which they belong. User modeling shell systems aim at the development of integrated representation, reasoning, and revision tools that form an "empty" user modeling mechanism to meet the requirements of generality, expressiveness, and strong inferential capabilities. When filled with application-dependent user modeling knowledge, these shell systems would fulfil essential functions of a user modeling component in an application system. User modeling shell systems will become part of the application after being filled with application-dependent user modeling knowledge. They receive information about the user from the application only and supply the application with assumptions about the user. User modeling servers are centralized user modeling components for more than one application. They have been developed using a client-server based architecture. User modeling servers are not integrated into an application, but communicate with the application through a network.

To support the adaptive Web site construction framework and assist diverse adaptive Web systems developed from this framework, we propose a generic user modeling server for adaptive Web systems (GUMSAWS) [47]. It is able to reach the goals of generality, extendability, and replaceability, and to satisfy the needs of the adaptive Web site construction framework. As a user modeling server, GUMSAWS communicates with adaptive systems through a network, and is a centralized user modeling

component for the adaptive systems developed from the adaptive Web construction framework.

1.2 Current Work

The architecture of GUMSAWS has been designed based on the previous work in the user modeling [47], and the implementation of adaptive systems based on the Adaptive Web Site construction framework. Some major components have been implemented, such as the User Model Description, the Group Model Description, the Authoring Tool, the Modeling Interface, the User Profile Manager, the Usage Group Handler, the Association Miner, the Inference Engine, the Recommender, and the Profile Editor. Machine learning and data mining techniques have been implemented and integrated into some of these components to learn and extract rules from users and user navigation history.

Currently, GUMSAWS contains the functions of building up a user/group profile, and storing, retrieving, updating and deleting entries. These functions are performed by system component (the User Profile Manager). GUMSAWS infers user property values through the Inference Engine by using diverse information sources, including direct information, groups information, association rules, and general facts. Direct information is the information directly provided by users and gathered through the User Profile Manager. Groups information is generated by the Usage Group Handler according to user navigation history. Association rules are extracted by the Association Miner from all users' property values. Through the Recommender, GUMSAWS provides its clients with recommendations according to user interests and association rules extracted by the Association Miner from the navigation history of users in the same group. GUMSAWS also provides the Profile Editor to allow users to see and modify their profiles.

GUMSAWS is implemented and examined within two application domains: E-tailer and E-news. The performance of inferring user property values from different information sources is evaluated by KDDCUP2000 data in the E-tailer domain. The example application in the E-news domain, PENS (Personalized Electronic News System), illustrates the basic user modeling functions provided by GUMSAWS and its user modeling task of providing recommendations based on user navigation history. PENS has been developed based on the adaptive Web site construction framework. It makes use of GUMSAWS to adapt to user navigation history, and to provide recommendations of news items strongly related to the news stories that users are currently reading.

1.3 Thesis Organization

This thesis is organized as follows:

Chapter 2 summarizes the results of exploring the user modeling area. The definitions of terms (user profile, user model, and user modeling) are given. Different types of user models and processes of user modeling are described. This chapter also briefly introduces user modeling history from user modeling components to user modeling shell systems and servers. Some examples of user modeling shell systems and servers are described as well.

Chapter 3 first gives the requirements and goals of system design for GUMSAWS. It then presents the GUMSAWS framework. This chapter also describes the five major subsystems of GUMSAWS, the Model Maintainer, the Information Source Generator, the Recommendation Provider, the System Repository, and the Model Description, and the interdependence of them. The way that GUMSAWS assists its clients (adaptive Web systems) is described as well.

Chapter 4 provides a detailed description of system components and the implementation of them. The implemented system components include the User/Group

Model Description, the Authoring Tool, the Modeling Interface, the User Profile Manager, the Usage Group Handler, the Association Miner, the Inference Engine, the Recommender, and the Profile Editor. The structure of implemented classes in some of these system components is presented. The implemented machine learning and data mining techniques, and other algorithms are also provided in this chapter.

Chapter 5 examines GUMSAWS in E-tailer domain to illustrate its user modeling task of inferring user property values. Experiments are carried out to evaluate the performance of inferring user property values from information sources, to compare the performance of inference by using the K-means algorithm and K-means+ algorithm to generate groups information, and to compare the performance of inference from different combinations of information sources. Experimental results are presented and discussed as well.

Chapter 6 describes how GUMSAWS can be made use of. It presents an example application that illustrates another user modeling task of GUMSAWS, providing recommendations based on user navigation history, and its basic user modeling functions.

Chapter 7 concludes the current work and proposes future work.

Chapter 2

User Modeling Overview

During the literature review, we explored the user modeling area by looking into different lines of research. The terms, user profile, user model, and user modeling, are defined and clarified. This chapter also introduces user modeling history from user modeling components to user modeling shell systems and servers, and describes some examples of user modeling systems and servers, including BGP-MS (Belief, Goal, Plan Management System) [29], GUMS (General User Modeling Shell) [12], UMT (User Modeling Tool) [7], um [21], LMS (Learner Modeling Server) [32], and Personis [22].

2.1 User Profile

A user profile is defined as a collection of information about a user, combining demographic information (name, age, location, to name a few), usage information (for example, pages visited, frequency of visit), and interests or goals (either explicitly stated by the user or implicitly derived by the system) [23]. The user profile is an instance of a user model for a particular user.

User profile data may be gathered from the client side, server side or from a proxy, either through direct interview or through observed behaviour such as purchases or

dialogue acts. The data may be categorized as demographic, behavioral, attitudinal or click stream data. There are really two types of user data: those that describe individuals and those that describe groups of users. Each individual user profile is based on contextual relevance observed during the user information access. The information may include demographics, goals and interests, browsing behaviour patterns, browsing capabilities, shopping behaviours, connection speed and type, and human relationships.

One of the key problems with user information is the difficulty in obtaining it. Another problem is the difficulty of verifying the veracity of the data. It has been suggested that users are neither interested in providing this information, nor are they necessarily even willing to provide it because of privacy concerns.

2.2 User Model

Most adaptive Web systems represent users via a user model. As defined by Alfred Kobsa [26], "user models are collection of information and assumptions about individual users (as well as user groups), which are needed in the adaptation process". To distinguish it from user profile, we define it as an abstract representation which contains explicit assumptions on all aspects of the user that may be relevant for the behaviour of the system. It represents both individual users and groups into which users are classified. A user model combines user preferences with the stated goals or interests and the behaviours performed by that user, and uses this information to deduce the perceived current goals and interests of the user. Systems build a user model for describing individual or group users and distinguishing them in order to provide different services for different users.

There are two main types of user models that may exist in adaptive Web systems [8]: the overlay model and the stereotype user model. An overlay model represents an individual user's information of each attribute defined for a user. For example,

AHA! 2.0 [10] keeps every concept and associated attribute in the domain model of the application into the overlay user model. However, an overlay model has the problem of initialization because of the difficulty of collecting detailed user information. Stereotype user model distinguishes several typical or "stereotype" users. It is simpler than the overlay mode and is generalized from overlay attributes. For example, MetaDoc [6] uses stereotypes (novice, begin, intermediate, and expert) to represent a user's knowledge. The problem with the stereotype model is that many efficient adaptation techniques require a more fine-grained overlay model. One way to solve this problem is to provide a mapping from a stereotype to an overlay model. Because of the problems of two types of models, it may be better to combine them in the following way: stereotype modeling is used at the beginning of work to classify a new user and to set initial values for the overlay model, then a regular overlay model is used. Many systems, for instance Web based adaptive education systems [43], use the combination of both two types of models.

Some other types of user models, for example, a Bayesian user models [17] and an episodic learner models [44], have been mentioned in some other works. They are developed based on either a special case or a specific technique.

2.3 User Modeling

User modeling is the whole process of constructing a user models and creating, updating or deleting user profiles. User modeling contains the functions which are to incrementally build up a user model, to store, update and delete entries in it, to maintain the consistency of the model, and to supply other components of the system with assumptions about user. There are three processes of user modeling [8]: collecting data about the user, processing the data to build or update the user model, and applying the user model to provide the adaptation. The last process is always performed by the adaptive applications because they are defined as the applications

which can provide automatic adaptation on the basis of the user model.

2.3.1 Methods of Collecting User Data

There are various ways to collect data from the user. The traditional way is to let the user provide information (for example, age, location, gender, occupation, income) directly by filling in a form. However, the user may provide untrue information or withhold information because of privacy issues. The data collected may not be reliable, and errors may be made when deducing a user's interests (for example) or to which group a user belongs.

Usage information is the information that can be tracked for observing users' behavior. It is perhaps the most important user data, and is extracted from a Web server log, which is the primary source of data in which the activities of Web users are captured. Usage information may be described in terms of simple page views, transactions (which are "significant" events, and may combine multiple page views), and sessions (which are combinations of page views or transactions that together represent users' behaviour). In addition to the simple sequence of events, information about time of access and frequency of access can also be captured as usage information. However, usage information is not reliable either. The page clicked by a user does not guarantee that the user attentively read its content.

To make user modeling simpler and more reliable, it is necessary to involve the user in the process of user modeling to get additional information from the user. For example, Barnes and Noble's E-tail site has a "add to wish list" link for each product to record the user's current or future goals or interests.

2.3.2 Techniques Used to Build User Model

Machine learning techniques are used to build a user model. Techniques used include linear models, TFIDF-based models, Markov models, neural networks, classification

and clustering techniques, rule induction techniques, and Bayesian theory-based techniques. Data mining techniques such as association rule mining and maximal frequent sequence mining [45], are also used when building user models.

The machine learning technique of the Self-Organized Map (SOM) is used by Murtaza et. al. to cluster users into groups [35]. Learning decision tree and association rule mining are usually used for rule induction (for example, [13] and [38]). Billsus and Pazzani [5] use the nearest neighbor algorithm to model user's short-term interest, and use a naive Bayesian classifier to model user's long-term interests. Bayesian theory-based techniques are also used by Lam and Mostafa [30] to model user interest shift. Zukerman and Albrecht introduced two new approaches, centent-based approach and collaborative approach, to build up a predictive statistical model [48]. They presented seven different machine learning techniques in achieving those approaches. Each of the techniques has unique features. The results from the comparison of these techniques indicate that the advantage of one over the other is domain dependent. Davison and Hirsh's study [9] shows that Markov model performs at least as well as the decision tree learned by C4.5 [39]. While in some other areas a decision tree learnt by C4.5 is a better choice over rules learnt by FFOIL [40].

2.3.3 User Modeling in Practice

In practice, many systems use various approaches and techniques to build up a user models. Techniques and approaches should be chosen according to specific cases. Some particular example systems are briefly introduced here.

The Lumiere project [17] focuses on providing assistance to computer software users by building a Bayesian user model, which is used to infer user's needs and goals based on user's background, action, and queries. The Lumiere project first classifies a user's actions and defines appropriate variables for those actions. It then uses Bayesian networks and influence diagrams to present the relationships between the actions and user's goals and to present probability distributions over these goals.

After that, user's goals are inferred based on the relationships and the probability distributions.

Syskill & Webert system [4] builds a set of profiles of user's interest in system-described topics. According to those profiles, the system then recommends pages which the user may be interested in. User's profile data is captured from the user's ranking a visited page with either "interesting" or "not interesting".

Web-EasyMath, a Web based algebra tutor system, uses a combination of stereotypes and the distance weighted k-nearest neighbor algorithm to initialize a student model. A student is interviewed the first time using the system. The system presents to the student a preliminary test to assess his/her knowledge level. The knowledge level is initialized by the stereotypes of novice, beginner, intermediate, and advanced. The system also calculates the distances between this student and other students in the same stereotype based on the student's performance in the preliminary test. Considering the calculated distances as weights, Web-EasyMath assesses the student's knowledge level of each particular concept of the domain and their proneness to make mistakes by using the distance weighted k-nearest neighbor algorithm.

SeAN, a server for adaptive news, is an adaptive system for the personalized access to news [3]. It personalizes both the selection of topics that are of interest to the user and the detail level of the presentation of each news item. Stereotype approach and Baye's theorem are used in this system for user modeling.

The COGITO project [1] is developed to improve the relationships between consumers and suppliers in e-commerce by employing "intelligent personalized agents". The main part of the system, user profile, is composed by two main frames: the frame of user data, which records the interaction data; and the frame of the user interests, which records the interests inferred by the system automatically.

Several approaches to create a user interest profile in the presence of positive evidence only are introduced in [41; 42]. Two machine learning techniques are used: a probabibilistic approach using a Bayes classifier and an instance-based approach

using a k-nearest neighbor algorithm.

2.4 User Modeling Components and Systems

Besides techniques used to build up a user model, there is another line of user modeling research, which focuses on generality of user modeling. It is getting more attention with the increased need of reusability. This section briefly introduces the history from user modeling components to user modeling shells, systems and servers.

In the early work, user modeling components were embedded into application systems, and were not distinct from the components that performed other tasks. These systems used various machine learning techniques to construct different types of user models. Example systems include SeAN [3] using Bayes theorem to build a stereotype model, the Syskill & Webert system [4] using a Bayesian technique called conjugate priors to build an overlay model, and a Web based adaptive education system [43] combining stereotype user model with overlay user model. The embedded user modeling components lack reusability and are only applicable to the adaptive systems that they belong to.

With the reusability of user modeling components gaining more and more attraction, "General User Modeling Systems" were named by Tim Finin in 1986. Later on, the term "user modeling shell systems" was used by Kobsa in 1990 [27]. User modeling shell systems aim at the development of integrated representation, reasoning, and revision tools that form an "empty" user modeling mechanism to meet the requirements of generality, expressiveness, and strong inferential capabilities. When filled with application-dependent user modeling knowledge, these shell systems would fulfil essential functions of a user modeling component in an application system. Some major user modeling shell systems for academic purpose have been developed including GUMS, UM, UMT, TAGUS, and BGP-MS [27]. Some of them will be described later in this chapter.

User Modeling shell systems become part of the application after being filled with application-dependent user modeling knowledge. They receive information about the user from the application only and supply the application with assumptions about the user. Many commercial user modeling shell systems have been developed using a client-server based architecture. They are not integrated into the application, but communicate with the application through a network. User modeling servers are centralized user modeling components for more than one application (possibly for all applications with which the user interacts) and seem to have the capabilities of domain-independent user modeling [24]. These commercial user modeling servers abstract user models from application systems, and build them as a user model server such that more than one application with a similar domain can access the information from it. A typical example of user modeling server is the Personalization Server. Other commercial user modeling servers include Group Lens, LikeMinds, Frontmind and Learn Sesame [27]. The current commercial user modeling servers are classified into four categories according to the methods they use [11]. One typical server from each category is selected, including 1) Group Lens, uses collaborative filtering approach; 2) Personalization server, uses simple production rules and stereotypes approaches; 3) Front Mind, uses simple production rules and Bayesian networks; and, 4) Learn Sesame, uses hierarchical clustering method.

2.5 Examples of Generic User Modeling Systems

This section introduces some generic user modeling systems and servers, including BGP-MS, GUMS, UMT, um, LMS, and Personis. A few issues, such as representation of user model, maintenance of user model, acquisition of user model, and so on, are mainly discussed to show the generality, expressiveness, and strong inferential capability of these systems and servers.

2.5.1 BGP-MS

BGP-MS (Belief, Goal, Plan Management System) is a user modeling shell (an independent software system) composed by many customizable application components [29]. It provides customized tools to clients for the purpose of adaptation based on assumptions about users' knowledge, beliefs and goals. One example client of BGP-MS is the KN-AHS [28], which is a hypertext system with the capability of dealing with the comprehension problems.

Briefly, BGP-MS offers a functional interface for updating the model of a user as well as some basic domain-independent inference mechanisms. More specifically, BGP-MS provides services including accepting observed user beliefs and goals from the client, sending the client interview questions that should be presented to the user, accepting user's answers to these questions, accepting observed user actions from the client, providing its current assumptions about the user to the client, and signaling important events in the user model to the client. Furthermore, a powerful representation language is defined in BGP-MS, which is the belief and goal description language (BGDL) for representing the observed user beliefs and goals.

BGP-MS represents user's beliefs and goals by using the partition mechanism KN-PART, including the representation of the domain knowledge, the individual user model, and stereotypes. However, within the partition, assumptions can be represented in the conceptual knowledge representation language SB-ONE (an example can be found in [25]) and first-order predicate calculus (FOPC). Domain knowledge is stored in a separate partition called SB, which stands for System Believes. Assumptions and information about the user are collected in other separate partition, such as assumptions of the system about the user's beliefs about the application domain, and so on. BGP-MS stores each stereotype in a separate partition. The partition will be linked to an appropriate partition of the individual user model if it applies to him/her. If the contents of two or more stereotypes overlap, partition hierarchies may optionally be used for constructing stereotype hierarchies. A simple example of

stereotype hierarchy can be seen in [28].

BGP-MS offers a set of pre-defined condition schemes which the application developer can appropriately instantiate to define the activation and retraction conditions of the stereotypes in her application domain. Frequently used schemes include IFKNOWN list, IFUNKOWN list, IFKNOWN% n, and IFKNOWN%OF n list. Instantiated schemes can be logically combined by the connectives AND, OR, and NOT, and also with any LISP code that returns a Boolean value. Each stereotype in BGP-MS provides a set of default beliefs represented in the underlying representation language.

The user model in BGP-MS consists of observations of the user's behaviour, of inferred data, and of stereotypical beliefs derived from active stereotypes. Through appropriate rules (which may, or may not, depend on the application domain), BGP-MS infers new beliefs from the concepts mentioned by the user. For example, if the user believes that concept C specializes concept B and that B specializes A, then BGP-MS infers that the user believes also that C specializes A. Periodically, after a certain (programmable) number of new beliefs have entered the user model, a reclassification process takes place, determining the stereotypes that should be activated and those that should be deactivated.

The main limitation of BGP-MS concerns the problem of resolving possible conflicts among predictions of different stereotypes or predictions that contradict information directly acquired from the user. The possible solution to this problem is some belief revision procedure possibly based on an assumption-based truth maintenance system that guarantees the consistency of the model.

2.5.2 GUMS

GUMS, a generic user modeling system, is implemented based on Prolog [12]. It is aimed at providing a set of services for the maintenance of assumptions about users' beliefs. GUMS does not draw assumptions itself. Instead, it accepts and stores new

facts about the user which are provided by the application system, verifies the consistency of a new fact with the currently held assumptions by trying to deduce the negated fact from the current assumptions, informs the application system about recognized inconsistencies, and answers queries of the application concerning its current assumptions about the user.

For each application, GUMS keeps a knowledge base of user models relevant to that application. Applications are responsible for acquiring information about the user and supplying it to GUMS to update the user model. The application must select the initial stereotypes for the user and add new facts about the user as it learns them.

The generality of a user model is formally defined in [20] with respect to three dimensions: the range of users, the forms of interaction, and the underlying system domain. A user model has interaction generality if it can be used with a variety of interaction modes, such as structured interactions or mixed initiative dialogue, and can be used with various modes of communication, such as natural language, menus, speech, and graphics. A domain general user modeling facility can be used with applications having a range of knowledge bases, such as diagnostic systems for medicine, mechanical devices, and electronic components. A general user modeling system must provide three essential facilities: representation and maintenance facilities for the contents of the model, access facilities for other components of the system or interface, and acquisition facilities for building the model.

Two of those main issues are discussed in detail in GUMS, including representation and maintenance of user model, and implicit user model acquisition. Representation and maintenance of information about the user is central to any user modeling activity. Representation of the user model focuses on how to represent user model by stereotypes, default rules, and failure as negation. Maintenance of the user model focuses on how to resolve conflicts between facts and rules.

For the representation of a user model, GUMS uses three default reasoning techniques to represent its beliefs about user knowledge: stereotypes, explicit default rules, and failure as negation. More specifically, a collection of stereotypes are organized into a taxonomy in GUMS. The individual models are installed in the stereotype hierarchy as leaves. GUMS uses a tree structure for stereotypes. Individual models are leaves in the tree. The root is a most general stereotype for users. Each stereotype can have a single subsuming stereotype, and a set (possibly empty) of subsumed stereotypes and a set (again, possibly empty) of individuals currently believed to be modeled by the stereotype. However, the stereotype system should form a general lattice, which means that an individual can have a set of subsuming stereotypes. The real contents of the stereotype however, consists of two databases of facts and rules, one definite and the other default. Therefore, GUMS has the shortage that an individual can only have a single stereotype. It means that there is only one choice left for the user model. Each stereotype is a collection of facts and rules that are applicable for any person who is seen as belonging to that stereotype. The GUMS system uses a very simple domain independent strategy for finding a new stereotype when the known facts about an individual contradict some facts associated with the user's current stereotype. The ancestors of the current stereotype are searched in order of specificity until one is found in which there is no contradiction.

The task of the maintenance facility is to update the individual user model, and restore consistency if necessary. Conflicts will be resolved. Several maintenance rules are defined in [12].

The acquisition problem in GUMS, as in most user modeling systems, is the need to explicitly encode large amounts of information about the potential system users. To model a user's beliefs about the system's domain, the task of building stereotypes may be more time consuming than building the domain knowledge base itself, because of the large number of stereotypes necessary. Furthermore, explicit model acquisition is error-prone due to the difficulty of classifying users and their likely beliefs. An

alternative to explicit user model acquisition is to build the model implicitly, as the user interacts with the system. Implicit acquisition avoids the explicit encoding bottleneck, and can reduce the burden on the application as well. GUMS uses implicit user model acquisition. User model acquisition rules are defined and described in [19]. The implicit acquisition rules rely on basic assumptions about the user and his/her behavior. The rules can be loosely partitioned into three categories: communicative rules, model-based rules, and human behavior rules.

Several problems, though, exist in the current GUMS system. One problem is that the stereotype system should form a general lattice, which means that an individual can have a set of subsuming stereotypes. Another problem is that GUMS does not extract all the available information from a new fact learned about the user because of the incompleteness of the truth maintenance system. Inefficiency is the third problem of the current GUMS system. GUMS does not record the results of inferences, but repeats the same inferences when the same query is posed again.

2.5.3 UMT

UMT [7], the user modeling tool, allows the user model developer to define user stereotypes that contain the characteristics of user subgroups in the form of attribute-value pairs. Stereotypes can be ordered in arbitrary hierarchies which support the inheritance of stereotype contents. Each stereotype possesses an activation condition which specifies when the stereotype can be applied to the current user. UMT also puts a rule interpreter at the disposal of the user model developer which allows for the definition of user modeling inference rules. Possible contradictions between assumed user characteristics also have to be explicitly defined using rules.

Stereotypes in the UMT system are organized in a multiple inheritance network through an IS-A relation defined by the inclusion relation holding between corresponding classes of users. Each stereotype is divided into two parts: a trigger, representing a sufficient membership condition of a special user with respect to the class denoted

by the stereotype, and the defaults, a list of attribute/values pairs representing typical traits of users belonging to the class. Each stereotype can be either active or non-active. Those active stereotypes that do not create contradiction will become the possible user models. Not all defaults of an active stereotype need to be included in the possible user models. In fact, defaults leading to an inconsistent user model are excluded.

Like GUMS, UMT does not draw assumptions itself, but accepts and stores new assertions about the user which are provided by the application system. Depending on the reliability of these assertions, they can be regarded as invariable premises or as (later still retractable) assumptions. Stereotypes that become activated due to new assertions about the user add still more assumptions, namely the attribute-value pairs describing the characteristics of the respective user subgroups. Some of the assumptions may possibly be contradictory. After stereotype activation, UMT applies all inference rules (including the contradiction detection rules) to the set of premises and assumptions, and records the inferential dependencies. A reason maintenance component in the system then determines all possible user models, which are all consistent sets of assertions containing the premises, a subset of the assumptions, and all assumptions derived from these premises and assumptions. The current user model will be selected from the set of possible user models using preference criteria (e.g. assumptions which were reported to the system by the application have a higher weight than assumptions from stereotypes). If inconsistencies with new information from the application are later detected or if the application disconfirms advice from UMT concerning the user, the assumptions on which the offending assertion was based can be detected (since inferential dependencies became recorded), and the set of possible user models will be revised and re-evaluated to find the new current user model.

The strengths of UMT lie in its mechanism for recording the inferential dependencies between assertions and for determining the most preferable maximally consistent

set of assumptions that can be maintained about the user at a given time. This mechanism is very time-consuming, however. Another possible obstacle to practical applicability are the limited representational and inferential abilities of UMT, which are based on attribute-value pairs and rules, respectively.

A problem which UMT shares with the current version of BGP-MS is the fact that stereotype retraction is not integrated into truth maintenance. The truth maintenance component can never decide to change the stereotypes that apply to the current user since stereotype activation and retraction is carried out by a completely independent component. An interesting aspect in the comparison between UMT and BGP-MS is the fact that inference is exclusively carried out in a forward-chaining fashion in UMT, while it is bidirectional in BGP-MS. If forward chaining is employed, processing occurs when observations about the user are communicated by the application, but not when queries of the application have to be answered. If backward chaining is employed, processing is deferred to the time when queries have to be answered. If the application and the user modeling component are one single process, both strategies are problematic since the operation of the application may be retarded by the inferences in the user modeling component, either at entry time or at query time. If the user modeling component operates concurrently with the application (as is the case in BGP-MS, but not in UMT), then forward-chaining seems advantageous since it can be performed while the application is idle. Unfortunately, though, the number of inferred data also has to be taken into account: it may well be that forward chaining generates numerous inferences that will not be needed, but which nevertheless have to be managed by the truth maintenance component. BGP-MS therefore allows the application developer to restrict the depth of forward chaining, so that a compromise between space complexity and time complexity can be found that is appropriate for the specific application domain.

2.5.4 um

um [21] is a toolkit for user modeling to represent assumptions about the user's knowledge, beliefs, preferences, and other user characteristics in attribute-value pairs from evidence, and to provide support for a variety of cooperative agents. The source of each piece of evidence, its type (observation, stereotype activation, rule invocation, user input, told to the user) and a time stamp is also recorded. The tools provided by um interpret representation of user models (either minimalist, basic or extended form); interpret and manipulate components (the basic element of a um model); and allow users to access and modify (if they wish) their user models.

Other systems, including BGP-MS, UMT and TAGUS, had interfaces for the use of the developer to scrutinise the models as they built and debugged them. However, the representation and reasoning mechanisms were not designed for the user to scrutinise them. The fundamental representation of a user model in and the viewer tools provided by the um toolkit give accessibility to user models. Two examples of um tools in use, the coaching system and the movie advisor, are described in [21] to show how um has been used. The generalized um architecture is provided as well.

Built upon the underlying representation of the user model provided by the um, the Personis [22] user model server provides generic scrutiny tools to enable the user to see and control their own user model. The views are the conceptual, high level elements shared between the server and each application. The views have an interaction with the design of the access control for the server.

2.5.5 LMS

A generic multi-agent architecture, the Pedagogical Agents Communication Framework (PACF), has been established for developing intelligent learning environments (ILE). As one of the main general agents, a Learner Modeling Server (LMS) [32] models learners and provides user models to several applications. It is configured by

the application developer, performs the acquisition and maintenance of the models, and keeps the models of learners in a database. The PACF is composed of several agents including interface agents, the LMS, the Server, and a tutoring agent. The LMS is a modeling agent in a multi-agent platform. The LMS consists of two tiers (the application tier and the database tier). Several modules exist in the application tie, including the database module, learner modeling module and communication module. The communications within the application tie are based upon Java Remote Method Invocation (RMI) protocol. The communication between the LMS and the applications is through a subset of the KQML language. A generic domain model is established to be used by the various applications. The parameterization of the LMS is through an authoring tool. Actions, weight values associated with actions, topics, application specific learning parameters and the classification and updating rules will be defined through the authoring interface as well. Basically, the LMS is able to serve applications which are all based on the same learning strategy and ontology.

2.5.6 Personis

The Personis user model server [22] focuses more on making adaptive systems scrutable: enabling users to see the details of the information held about them, the processes used to gather it and the way that it is used to personalize an adaptive hypertext. The systems to be moved out of the laboratory have to meet legal requirements such as the European Community Directive on Data Protection. It is in the spirit of such legislation that users be able to access and control their own data. Other problems addressed in Personis include reuse of user model information across applications, the need for ensuring the user's privacy, control and ability to scrutinise their user model and the processes for personalization.

The Personis user model server provides generic scrutiny tools to enable the user to see and control their own user model. The views are the conceptual, high level

elements shared between the server and each application. The views have an interaction with the design of the access control for the server. Personis allows the user to define just which applications are allowed to see each part of the user model. The user can also control the information sources that should be made available to each application.

2.6 Concluding Remarks

The terms user profile, user model, and user modeling, are defined and clarified in this chapter. A user profile is defined as a collection of information about a user, and is an instance of user model for a particular user. User model is an abstract representation which contains explicit assumptions on all aspects of the user that may be relevant for the behaviour of the system. User modeling is the whole process of constructing a user model and creating, updating or deleting user profiles. It consists of two processes, collecting data about a user and processing the data to build or update the user model. Methods of collecting data about a user and problems of them are described. Machine learning and data mining techniques that are used to build a user model, and some example systems are introduced as well.

User modeling history from user modeling components to user modeling shell systems and servers is introduced. User modeling components are embedded in application systems, and lack reusability. User modeling shell systems form an "empty" user modeling mechanism. They become part of the application after being filled with application-dependent user modeling knowledge. They receive information about the user from the application only and supply the application with assumptions about the user. User modeling servers are centralized user modeling components for more than one application. They assist more than one applications concurrently in a similar domain.

Some examples of generic user modeling shell systems and servers are described.

A few issues, such as representation of user model, maintenance of user model, acquisition of a user model, are mainly discussed to show generality, expressiveness, and strong inferential capability of these systems and servers. The advantages and disadvantages of these systems are described as well.

Chapter 3

A Generic User Modeling Server

GUMSAWS is designed according to the requirements and goals presented later in this chapter. The framework of GUMSAWS consists of five major subsystems: the Model Maintainer, the Information Source Generator, the Recommendation Provider, the System Repository, and the Model Description. This chapter presents the GUMSAWS framework, and briefly describes its major subsystems and components, and their interdependence. It also describes the interactions between GUMSAWS and its clients, between users and adaptive Web systems, between users and a generic interface, and between authors and GUMSAWS.

3.1 Requirements and Goals

The GUMSAWS framework design is based on the requirements of offering basic user modeling functions, and of being a user modeling server. There are also several general goals which describe performance measures about the framework, including generality, extendability, and replaceability.

3.1.1 Design Requirements

As a generic user modeling server, GUMSAWS has to be designed to provide basic user modeling functions, to have the capabilities of domain-independent user modeling, and to communicate with adaptive Web systems through a network. Functions of user modeling include incrementally building up a user model, storing, updating and deleting entries, maintaining the consistency of the model, and supplying other systems with information about users. GUMSAWS should be applicable to adaptive Web systems in the same domain, and be configurable across different domains. As a server, the communication between GUMSAWS and adaptive Web systems should be through a network. A new protocol for communication may be necessary.

As pointed out, it is difficult to obtain user information because users are neither interested in providing their information, nor are they necessarily even willing to provide it for privacy concerns. Therefore, GUMSAWS should be able to infer users' missing property values from the existing user information and other observed information sources. GUMSAWS should also be able to provide assumptions about users' goals and plans from observed users' behavior, provide a well-structured database for storing user profiles, and accept queries from its clients and provide them with users' information for the purpose of adaptation.

3.1.2 General Goals

There are several general goals which describe performance measures about the framework.

- Generality: The framework should be applicable to multiple application domains. It should act as a centralized user modeling component for more than one application (possibly for all applications with which a user interacts) and has the capabilities of domain-independent user modeling.

- Extendability: The framework definitions should allow extension, so that new definitions can be developed from other fundamental definitions. For example, new information sources can be generated for the purpose of inference by adding a new type of miner. It should be able to define new fundamental objects within the system, although this is likely infeasible.

- Replaceability: Diverse machine learning and data mining techniques should be implemented in the system, and they should be replaceable by other techniques. For example, for the purpose of generating groups of users, the K-means+ clustering algorithm is implemented to group users with common interests. Other clustering algorithms, such as K-means, should be able to replace K-means+ and to be integrated into the system. Other replacement opportunities should also exist, for example, the Apriori algorithm implemented to mine association rules could be replaced by the AprioriTID algorithm.

3.2 Interaction Architecture

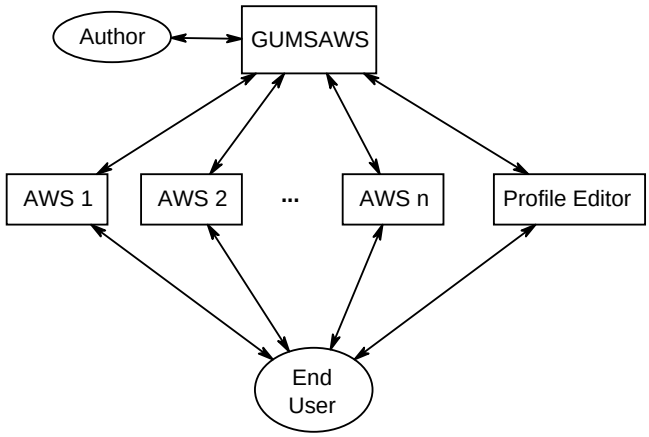


Figure 3.1: Interaction Architecture

Like other user modeling servers, GUMSAWS acts as a centralized user modeling component and interacts with more than one adaptive Web systems (AWS 1, AWS

2, ..., and AWS n in Figure 3.1) at the same time. It captures user information from its clients, and builds user models. It can support reuse of the user models over all these clients.

User information may be provided through the adaptive Web systems during users' interactions with these adaptive Web systems. It may also be provided through a generic interface called the Profile Editor. This interface makes adaptive Web systems transparent. It enables users to see information held about them, and to modify their profiles. Requests sent through the interface will be handled directly by the server.

In the proposed architecture, the author is responsible for defining a user model and group model for the adaptive Web systems. The author has to be familiar with the current domain that these adaptive Web systems belong to. The author defines concepts and relationships between users existing in the adaptive Web systems.

3.3 Framework

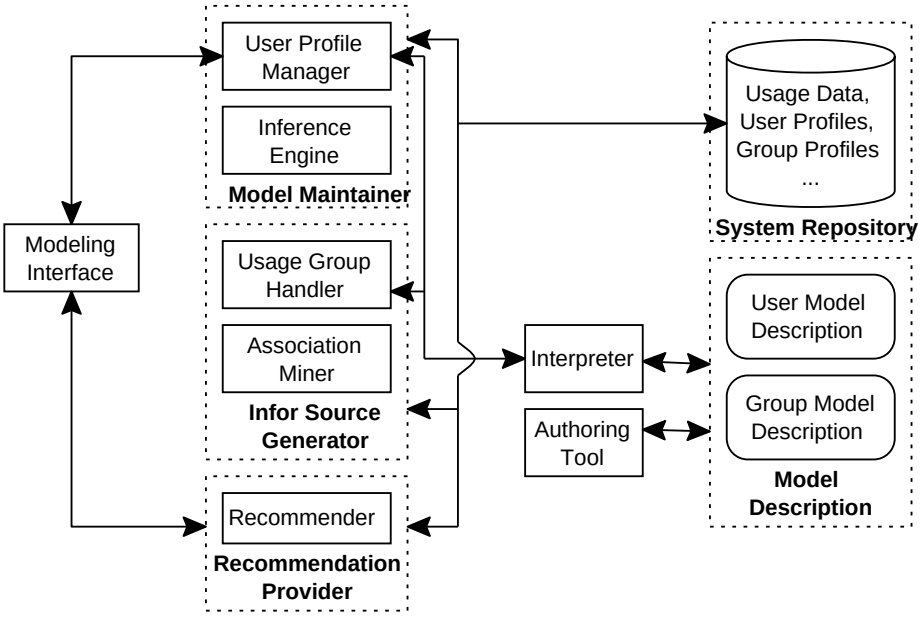


Figure 3.2: GUMSAWS Framework

Figure 3.2 illustrates the GUMSAWS framework. In this figure, files are represented as rounded rectangles, databases are represented as columns, engines or components in the system are represented as rectangles, and arrows represent data flow between system components or between system components and data repositories. System components are grouped into five sub-systems, which are represented as dashed rectangles. These five sub-systems are:

1. The Model Description (MD): Consists of two components, the User Model Description and the Group Model Description. These are data repositories that store domain-dependent Intermediate Format Vocabularies (IFVs) and application-dependent description for user and group models. The IFV is the schema in intermediate format for describing concepts and relationships related to an individual user or group of users existing in adaptive Web systems.

2. The Model Maintainer (MM): Offers a variety of user modeling functions, including the function of building up a user/group profile, the function of storing, retrieving, updating and deleting entries, and the function of inferring user property values. Two components, the User Profile Manager and the Inference Engine, are grouped into this sub-system.

3. The Recommendation Provider (RP): Is used to provide recommendations according to information sources about user's interests and of discovered association rules from visiting history. Although only one system component, the Recommender, is included in this sub-system, many other recommendation components can also be implemented and integrated into this sub-system.

4. The Information Source Generator (ISG): Generates the information sources of groups information and association rules for the MM to update user's property values and the RP to provide recommendation to users. Two components, the Usage Group Handler and the Association Miner, are included into this sub-system. User groups are generated by the Usage Group Handler according to

users' visiting history. Two types of association rules are discovered by the Association Miner. One type of association rules is amongst all properties of all existing users, which indicates that given a particular user property value, which other user property values also existed. Another type of association rules is mined from visiting histories of the users in the same group, which indicates that given a particular page read by users in a group, which other pages the users have also read.

5. The System Repository (SR): Is used to store usage data, user profile data, and group profile data. It also stores information sources of direct information, groups information, association rules, and general facts. The SR also connects some system components together. It is constructed in the initialization stage by some system components, such as the User Profile Manager, the Usage Group Handler, and so on. It will be updated by them as well during the system activity.

The Modeling Interface (MI) is an interface between adaptive Web systems and GUMSAWS. The main function of the MI is to forward adaptive systems' requests to components of GUMSAWS. The communication between the MI and the sub-systems is through a network. The communication between the sub-systems within GUMSAWS may be either through a database or other system components. For example, the MM and the ISG access the MD through the Interpreter; and both of these two sub-systems do not communicate with each other directly, but only through the SR.

Besides the MI, two other system components are not grouped into any sub-systems. They are the Interpreter and the Authoring Tool (AT). However, they provide connection between sub-systems or provide the interface for interactions between the author and the system. The Interpreter provides the functions of interpreting the model description and creates database tables to store user/group profile data and the

default user/group profile. The Interpreter is also responsible for providing the MM with the user model structure to operate on user profiles, and the ISG with the group model structure to operate on group profiles. The AT accesses model description and provides interface for authors to specify the application-dependent user/group model description. Through the AT, the author may define user and group models, and default user and group profiles.

Chapter 4

System Implementation

System components (except the System Repository, and the Model Description) are implemented by using Java. Java provides sufficient APIs for database connection, network programming, and GUI programming. We also choose Eclipse as the integrated development environment of Java because it is free and user friendly, and offers some useful plug-ins, like XMLBuddy.

The System Repository is developed using a MySQL database server, which is the most popular open source database [36]. MySQL database server is also chosen because it is free, extremely fast, and easy to customize. In order to access a MySQL database through Java code, we use the JDBC connector (Java DataBase Connector). JDBC technology is an API that provides cross-DBMS connectivity to a wide range of SQL databases.

Both the User Model Description (UMD) and the Group Model Description (GMD) are implemented using the Resource Description Framework (RDF). Interpretation of them is implemented using Jena, which is a Java framework for building Semantic Web applications. Jena provides a programmatic environment for RDF.

The proposed system has been implemented. The major classes of some implemented system components are presented in the Appendix.

This chapter will provide detailed description of the system components and their

implementation. Although not all functions of each Java class in the system components will be described in detail, the classes of system components and the relationships among them will be introduced. The implementation of machine learning and data mining techniques (K-means, K-means+, and the Apriori algorithm) will also be described.

4.1 User and Group Model Description

The UMD and GMD provide intermediate format vocabularies (IFVs) for different domains to describe a user model and a group model, and specify default user and group profiles. The IFV is the schema in intermediate format for describing concepts and relationships existing in adaptive Web systems (in our case, mainly between users or within user's properties). Two IFVs are developed in our system, including the user model vocabulary and group model vocabulary for the E-news domain. The user model vocabulary (UMV) is used for describing application-dependent user model, and the group model vocabulary (GMV) is used for describing application-dependent group model.

Both IFVs and user/group model are described using the RDF serialized in XML. This section will provide brief introduction to RDF, the detailed explanation of IFVs and application-dependent user and group models, and the UMD and the GMD for the E-news domain.

4.1.1 Resource Description Framework

The Resource Description Framework (RDF) is developed by the W3C for Web-based metadata using XML as an interchange syntax. It can be used to represent information about things (including human beings), even when the things cannot be directly retrieved on the Web. It provides inter-operability between applications that exchange machine-understandable information on the Web. RDF emphasizes

facilities to enable automated processing of Web resources. A document provided by W3C to describe RDF specifications and provide examples of using RDF can be found in [34]. Here, we provide brief introduction to the core of RDF and its basic principle.

RDF identifies resources using Web identifiers called Uniform Resource Identifiers (URIs). It is a Web standard, and uses property values to describe web resources. It also uses the the terms subject, predicates and objects. At the core of RDF we have the RDF Data Model for representing named properties and their values. These properties serve both to represent attributes of resources (and in this sense correspond to usual attribute-value pairs) and to represent relationships between resources. The RDF data model is a syntax-independent way of representing RDF expressions. The RDF Syntax is for expressing and transporting this metadata in a manner that maximizes the inter-operability of independently developed web servers and clients. The syntax uses the eXtensible Markup Language (XML). RDF schemas are a collection of information about classes of RDF nodes, including properties and relations. RDF schemas are specified using a declarative representation language influenced by ideas from knowledge representation, e.g., semantic nets, frames, predicate logic, as well as database schema representation models such as binary relational models, and graph data models. RDF in itself does not contain any predefined vocabularies for authoring metadata. It is however expected that standard vocabularies will emerge. After all this is a core requirement for large-scale inter-operability. Anyone can design a new vocabulary, the only requirement for using it is that a designating URI is included in the metadata instances using this vocabulary.

4.1.2 User Model Description

The UMD is part of the MD, including the UMV and the description of application-specific user model with the default user profile. The UMV is domain dependent but application independent. Various UMVs for their corresponding domains will be

specified in the system and will later be used to define the application-specific user model with its default user profile.

```
<rdfs:Description rdf:ID="UserModel">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class" />
  <rdfs:comment>User Model is a collection of properties about user</rdfs:comment>
</rdfs:Description>
...
<rdf:Property rdf:ID="dgmodel">
  <rdfs:domain rdf:resource="#UserModel" />
  <rdfs:range rdf:resource="#DGModel" />
  <rdfs:comment>demographic information about the user</rdfs:comment>
</rdf:Property>
...
<rdfs:Class rdf:ID="DGModel">
  <rdfs:comment>Model that represents demographic information about user</rdfs:comment>
</rdfs:Class>
...
<rdf:Property rdf:ID="age">
  <rdfs:domain rdf:resource="#DGModel" />
  <rdfs:comment>User's age is represented by a interger number</rdfs:comment>
  <rdfs:range rdf:resource="&xsi;integer" />
</rdf:Property>

<rdf:Property rdf:ID="gender">
  <rdfs:domain rdf:resource="#DGModel" />
  <rdfs:comment>User's gender is represented by either "male" or "female".</rdfs:comment>
  <rdfs:range>
          <rdf:Alt>
                  <rdf:li>male</rdf:li>
                  <rdf:li>female</rdf:li>
          </rdf:Alt>
  </rdfs:range>
  </rdf:Property>
...
```

Figure 4.1: User Model Vocabulary for E-news Domain

4.1.2.1 User Model Vocabulary

The UMV contains description of concepts and relationships for all (as complete as possible) user properties in a specific domain. User properties are described in different classes, to which they belong. A property name and its value range are specified as well. As presented in Figure 4.1, 'DGModel' is defined as a class to represent demographic information about users. It consists of, for example, properties of age and gender. User's age is represented by an integer number. The value of

```
<rdfs:Class rdf:ID="InterestedTopics">
  <rdfs:comment>indicates how users are interested in predefined topics</rdfs:comment>
</rdfs:Class>

<rdf:Property rdf:ID="NATIONAL">
  <rdfs:domain rdf:resource="#InterestedTopics" />
  <rdfs:comment>User's interests in national news.</rdfs:comment>
  <rdfs:range rdf:parseType="Literal" />
</rdf:Property>

<rdf:Property rdf:ID="WORLD">
  <rdfs:domain rdf:resource="#InterestedTopics" />
  <rdfs:comment>User's interests in world news.</rdfs:comment>
  <rdfs:range rdf:parseType="Literal" />
</rdf:Property>
...
```

Figure 4.2: Interested Topics Class in User Model Vocabulary

user's gender may be either 'male' or 'female'. Another class called 'InterestedTopics' (shown in Figure 4.2) is defined as the class to represent users' interests in different news categories in the E-news domain, such as national news category and world news category.

The UMV will be used for defining the application-specific user model and default user profile. The concepts and relationships defined in UMV are also accessible by other parts in the Model Description.

4.1.2.2 User Model

For adaptive systems with the same mechanism in the same domain, an application-specific user model will be defined based on the UMV predefined for this domain. A set of appropriate user properties will be selected to describe the user model according to the needs of the application. As presented in Figure 4.3, user's age, gender, education level, education field are used to describe the user's demographic information. Users' interests in different news categories are also presented in the figure. The default user profile will be specified as well. For instance, default value for gender is chosen as 'male'. The selected user properties and their relationships will be interpreted by the Interpreter and used for constructing a database structure to store user profiles and

```
<umdv:UserModel rdf:ID = "defaultUserProfile">
        <umdv:dgInfor rdf:resource = "#dgInfor"/>
        <umdv:interestedTopics rdf:resource = "#interestedTopics"/>
...
</umdv:UserModel>
...
<umdv:DGModel rdf:ID = "dgInfor">
        <umdv:age>28</umdv:age>
        <umdv:gender>male</umdv:gender>
        <umdv:educationLevel>bechalor</umdv:educationLevel>
        <umdv:educationField>Computer Science</umdv:educationField>
...
</umdv:DGModel>
<umdv:InterestedTopics rdf:ID = "interestedTopics">
        <umdv:FINANCIAL>0</umdv:FINANCIAL>
        <umdv:ACADEMIC>0</umdv:ACADEMIC>
...
</umdv:InterestedTopics>
...
```

Figure 4.3: Description of User Model

the predefined default user profile. The default user profile will be assigned to a new user without any observed information and action. The user profile will be updated when more user information is observed by the system.

4.1.3 Group Model Description

Groups represent clusters of users who have properties in common. They are represented by their centers (results from clustering process). Centers are essentially the set of average values for some features within groups. Users will be classified into a group according to how close they are to the centers of the groups. Each group is identified by a unique name (group ID).

Similar to the UMV, the group model vocabulary (GMV) contains a description of concepts and relationships for all (as complete as possible) group properties in a specific domain. As shown in Figure 4.4, properties may include the 'groupID' indicating a group, and the 'academic' property representing a feature for clustering.

The description of group model, shown in Figure 4.5, represents the group model for the adaptive systems with same clustering mechanism in the E-news domain, and

```
<rdfs:Description rdf:ID="GroupModel">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class" />
  <rdfs:comment>Group Model is a collection of properties about a group of users</rdfs:comment>
</rdfs:Description>

<rdf:Property rdf:ID="groupID">
  <rdfs:domain rdf:resource="#GroupModel" />
  <rdfs:comment>GroupID uniquely identifies each group by a number.</rdfs:comment>
  <rdfs:range rdf:resource="&xsi;integer" />
</rdf:Property>

<rdf:Property rdf:ID="academic">
  <rdfs:domain rdf:resource="#GroupModel" />
  <rdfs:comment>Interest value of academic category is a float number</rdfs:comment>
  <rdfs:range rdf:parseType="Literal" />
 </rdf:Property>
...
```

Figure 4.4: Group Model Vocabulary for E-news Domain

```
<gmdv:GroupModel rdf:ID = "defaultGroupProfile">
        <gmdv:groupID>-1</gmdv:groupID>
        <gmdv:groupPopu>0</gmdv:groupPopu>
        <gmdv:newPopu>0.0</gmdv:newPopu>
        <gmdv:financial>0.0</gmdv:financial>
        <gmdv:academic>0.0</gmdv:academic>
        <gmdv:social>0.0</gmdv:social>
        <gmdv:people>0.0</gmdv:people>
        <gmdv:misc>0.0</gmdv:misc>
</gmdv:GroupModel>
```

Figure 4.5: Description of Group Model

the default group profile.

4.2 Authoring Tool

In the initialization stage, the author needs to define the user models and group models and specify default user and group profiles according to the IFVs, the user model vocabulary and the group model vocabulary. The Authoring Tool (AT) component provides the author with the Graphical User Interface (GUI) to interact with the system. The GUI is created using Java Swing components.

A screen shot of the AT is shown in the Figure 4.6. Through the GUI, the author

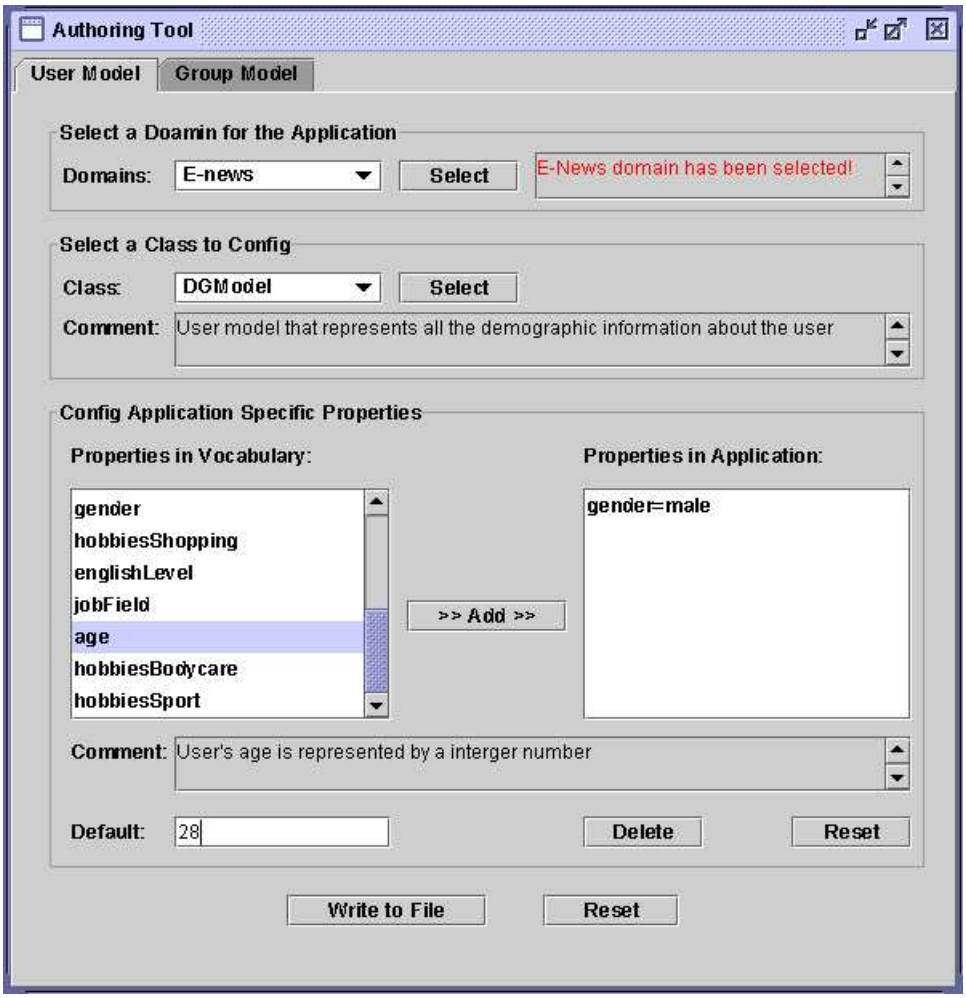


Figure 4.6: The Authoring Tool

can select one IFV (for example, the IFV for E-news domain in the screen shot) from one of the predefined IFVs for different domains. Each class defined in the IFV can be chosen by the author to specify user properties within this class. At the same time, the default value for a property can be specified through the 'default' text field. Specified user models for applications can be viewed and changed before being written into data repositories.

4.3 Modeling Interface

The Modeling Interface (MI) is an interface between adaptive Web systems and GUMSAWS. The main function of the MI is to forward the adaptive systems' requests to appropriate instructions (identified by a unique ID called request ID) provided by components of GUMSAWS (the User Profile Manager and the Recommender). The MI receives a request from a client and forwards it to one of the components, then awaits a reply to forward it to the client. The interface can handle concurrent requests. It means that clients can send multiple requests to the MI in a row and collect the results afterwards. All communications between clients and the interface and also between the interface and system components would be done through the User Datagram Protocol (UDP). UDP offers non-guaranteed datagram delivery and gives applications direct access to the datagram service.

The MI is implemented as a set of concurrent threads. Beside the shell class that contains the main function, there are two permanent threads: the registration thread and the message relay thread. For each incoming request from the adaptive systems, the message relay thread spawns a new thread in order to take care of the request and the corresponding reply. This thread will be discarded if the response is forwarded to those clients or a timeout occurs. The interface maintains a table containing all pairs of [scID, PointOfService]. This table is used for mapping between a system component ID and its point of service.

4.3.1 Registration

The MI locates every registered system component so that the requests from clients can be delivered to the proper system component and the responses can be returned to the clients correctly. For this purpose, each system component is supposed to declare its point of service (IP address + UDP server port) to the MI upon startup. System components are recognized by their scID. For registration, a system component sends

a packet to the MI at UDP port 4444 in the format presented in Table 4.1.

Table 4.1: Format of Registration Packet

2 bytes	4 bytes	2 bytes
scID	SC IP Address	SC Port Address

In the registration packet, the IP address comes from high byte to low byte. For example, if system components are located on a machine with IP 192.168.1.100, then the first byte after scID would be 192. For two other fields, the high byte comes first as well. If there is no reply from the MI, this indicates a successful registration. A system component can register one or more times, at any time.

4.3.2 Message Relaying

The MI forwards all incoming requests from a client to the respective system component and sends the received response back to the client. Each system component is supposed to reply to a request no later than a time limit (currently set to 10 seconds). If the timeout happens, the MI will no longer accept the response from this system component. It is possible that a system component receives a few requests in a row. Therefore, each system component is supposed to be ready immediately to receive the next request even if the previous request is not yet replied to. The format of a request packet from the MI to a system component is presented in Table 4.2.

Table 4.2: Format of Request Packet from MI to System Component

2 bytes	1 byte	Variable length
Sequence Number	Request ID	String of parameters

The sequence number is a serial number that each client's process uses to keep track of the requests already dispatched. The request ID specifies the service that the client has asked the system component to accomplish. If the accomplishment of service requires some parameters from the client, they come serially at the end of message. The format of the list of parameters is predefined. The phases in the

list of parameters are separated by the symbol '#'. The property name and its value (optional) are separated by the symbol '&'. The given example can be 'default#password&default', which means the user name of 'default' and the password of 'default'. Another example can be 'default#language', which means the user name of 'default' and this user's property of 'language'. The format of response from the system component, back to the same port of the MI that the request came from, is presented in Table 4.3.

Table 4.3: Format of Response Packet from System Component to MI

2 bytes	Variable length
Sequence Number	Response

The system component should repeat the sequence number of corresponding request packet at the header of reply. The MI forwards the response to the client intact, to the same port that the original request came from. The client uses the format presented in Table 4.4 to send a request to the MI, at port 5555.

Table 4.4: Format of Request Packet from Client to MI

2 bytes	2 bytes	1 byte	Variable length
scID	Sequence Number	Request ID	String of parameters

The scID indicates that this packet goes to which system component. This is the only field that the MI actually interprets. The high-byte of scID comes first. If the client sends a packet to the MI with an unknown scID (a scID that has not yet been registered with the MI), it simply drops the incoming packet and no error message will be returned.

4.4 User Profile Manager

The User Profile Manager (UPM) is responsible for providing adaptive Web systems with information about users and their navigation patterns. The UPM is also in

charge of instantiating users' profiles from the user model and the default values, and later on, updating the profile according to directly provided information from users. In the initialization stage, through the Interpreter, the UPM reads the user model description, creates database tables for the user model, and inserts default values into the database. It also provides a list of services, such as checking the existence of a user, creating and deleting a user profile, and updating and retrieving a property. Detailed information about the services provided by the UPM is summarized in Table 4.5. When the UPM receives a request from the MI, it will execute one of the instructions according to the request ID (reqID in the table) and the list of parameters attached in the received message.

Table 4.5: List of Services Provided by the UPM

Instructions	reqID	Parameters	Description
exist	1	userID	check existence of a user
check	2	userID, password	check user's login
create	3	userID	create a new user
delete	4	userID	delete a user
update	5	list of properties and values	update property value
retrieve	6	list of properties	retrieve property value
reset	7	list of properties	reset property value as default
addValue	8	list of properties and values	insert a set of property values
checkValue	9	list of properties and values	check validity of properties and their values

4.4.1 Class Diagram

The UML class diagram for the UPM is shown in Figure 4.7. The CTupm class (See Appendix A.1.1) registers the UPM to the MI first, and then launches a UPManager thread for accepting requests from the MI. This class also defines the global variables for the database connection, and the IP address and port number of the computer that runs the MI, and specifies the service port for the UPM (6666). The UPManager

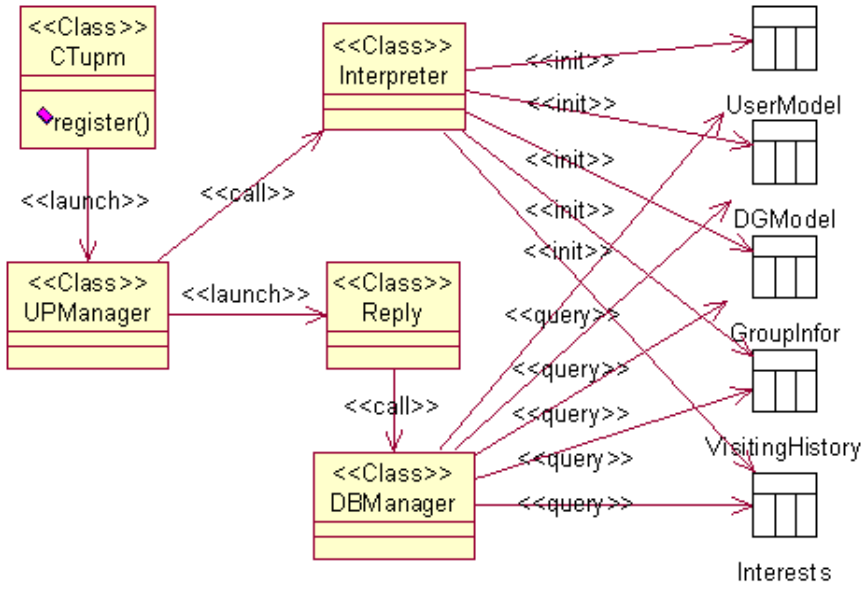


Figure 4.7: UML Class Diagram for the UPM

class (See Appendix A.1.2) calls functions provided by the Interpreter class to create a vector storing user model structure in memory, to create database tables for user model, and to insert the default profile into the database. This class also accepts requests from the MI and launches a new Reply thread to handle each request. The Reply class (See Appendix A.1.3) interprets the request passed from the UPManager and reads the request ID attached in the request. The request ID will determine which service should be used for handling the request. The generated response from the service will be sent back to the MI. The DBManager class contains functions of querying the database. Those functions are called by the Reply class to handle each request.

4.5 Usage Group Handler

The Usage Group Handler (UGH) is responsible for generating user groups. Like the user model, the group model and the default values are described in RDF format based on the predefined Group Model Vocabulary. The UGH reads the description

of group model and creates database tables for storing user group information. The UGH then groups existing users together according to their interests, and assigns a user into an existing group based on the evaluation of the user's distance from the groups' centers. Users' interests are extracted from their visiting history and are represented by the number of pages that users have read in each category. Moreover, the earlier the page has been read, the less weight it will have in the category that the page belongs to because users' interests might change over time. Therefore, a user's interest is represented by a vector of which each element is the number of pages that the user has read in each category. The UGH determines how close the user is to each group center by calculating the Euclidean distance between the vector of the user's interest and the vector of each group center, which is the mean of all users' interest vectors in the group. Finally, the user will be assigned to the group whose center is the closest to the user.

4.5.1 Clustering Algorithms

Two clustering algorithms, K-means and K-means+, are used in the implementation of the UGH for the purpose of grouping.

Algorithm 1 K-means Algorithm

1. Randomly select k different user vectors to represent the initial centers of user groups;
2. Assign each user to the group that has the closest center to the user.
3. When all users have been assigned, recalculate the k group centers.
4. Repeat Steps 2 and 3 until the group centers no longer move.

The standard K-means algorithm [33] clusters users into k groups, where k is the predefined number of cluster centers. As presented in Algorithm 1, the clustering procedure follows a simple and easy way to assign a given user to one of the k clusters.

Another clustering algorithm is the K-means+ algorithm (Shown in Figure 4.8).

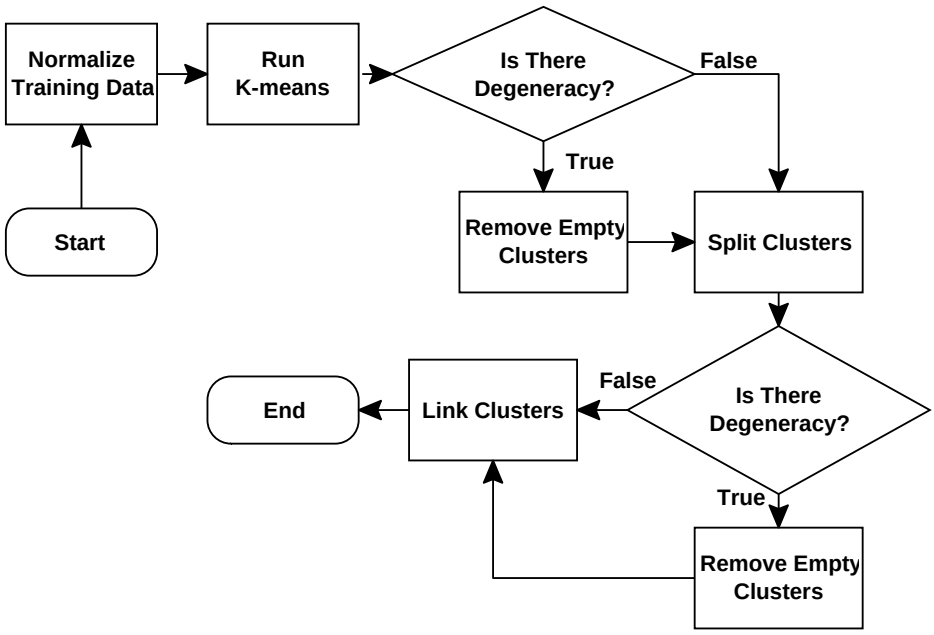


Figure 4.8: K-means+ Algorithm

The K-means+ algorithm is developed based on the standard K-means algorithm [15; 16]. The K-means+ algorithm determines a semi-optimal number of clusters automatically, and the number of initial centroid seeds not critical to the cluster result. The most distinguished feature of the K-means+ algorithm is that the number of clusters can be self-defined according to the statistical nature of data. In the K-means+ algorithm, three additional processes are introduced, the process of eliminating degeneracy, the process of splitting clusters with outliers, and the process of linking close clusters. As a result, the number of clusters can be automatically adjusted according to the distribution and density of the data. More specifically, different from the K-means algorithm, of which the number of clusters (k) remains fixed, the K-means+ algorithm adjusts the value of k autonomously by exploiting the statistical nature of the data. Additionally, the K-means+ algorithm also overcomes the shortcoming of degeneracy by deleting empty clusters. Compared with the K-means algorithm, the K-means+ algorithm uses the multi-centered clusters to obtain better performances.

4.5.2 Class Diagram

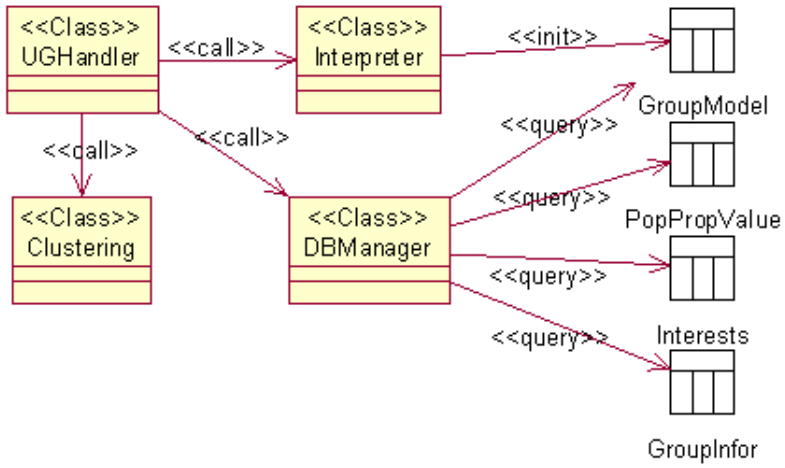


Figure 4.9: UML Class Diagram for the UGH

The UML class diagram for the UGH is shown in Figure 4.9. The UGHandler class (See Appendix A.2.1) calls functions of the Interpreter class to initialize the UGH, including creating a vector to store group model structure in memory and creating database tables for the group model. This class also launches a thread to retrieve users' interests from the database, and to call the functions from the Clustering class to group the existing users together and update their group information after a period of time. The two clustering algorithms are implemented in the Clustering class to offer the grouping functions. The DBManager class (See Appendix A.2.2) contains functions for querying the database. Those functions are called by the UGHandler class to retrieve and update user and group information.

4.6 Association Miner

The Association Miner (AM) is responsible for providing information sources of association rules. It mines association rules from either the users' demographic information or navigation history. Hence, two types of association rules can be discovered.

One type of association rule is amongst all properties of all users, which indicates that given a particular user property value, which other user property values also existed. These association rules will be used to infer the user's property values according to their current property values. Another type of association rule is mined from visiting histories of the users in the same group, which indicates that given a particular page item read by users in a group, which other pages the users have also read. These rules will be used to provide users with recommendations according to their current reading page. The support and confidence values of the association rules must be above a threshold to be considered. The extracted rules are ranked based on a measure that is calculated from the support and confidence values.

4.6.1 Apriori Algorithm

Given a finite set of items called itemset, an association rule is written as $x \rightarrow y$, where x and y both are the subsets of the itemset. The term of $x \rightarrow y$ should be read as x means or implies y. It tells us about the association between two or more items. The association rule has support s, which means $s\%$ of itemsets contains x and y. The support is the frequency of itemsets that contains both x and y. The rule holds confidence c, which means c of itemsets containing x also contain y. Confidence c can be calculated as the support of itemsets containing x and y divided by the support of y. Based on an example of buying products in a supermarket, we assume that the itemset is the set of all the products that a customer bought. We suppose that one of the association rules is $apple \rightarrow orange$. This association rule has the support s, which means $s\%$ of product set bought by the customer contain apples and oranges. It holds the confidence c, which means $c\%$ of products set bought by the customer contain apples also contain oranges.

Association rules are generated by applying the Apriori algorithm [18; 2]. The Apriori algorithm first finds out all the frequent itemsets. It will throw away many

candidate itemsets by the idea that every subset of a frequent itemset is also frequent. Thus, all larger candidate itemsets must be built on the itemsets that have frequency over minimum support. The process of finding out all the frequent itemsets is presented in Algorithm 2.

Algorithm 2 Algorithm of Finding Support of Itemsets

a) Find all single items whose frequency is above minimum support;
b) For all supported single items, find all pairs whose frequency is above minimum support;
c) For all supported pairs, find all triples whose frequency is above minimum support;
d) Continue, till no itemset can be found.

After the itemsets whose supports are larger than the minimum support have been found, association rules need to be generated from all those frequent itemsets. For each itemset l that contains more than one item, all non-empty subsets x of it should be found first. For every x, a rule of $x \rightarrow (l - x)$ can be generated if the ratio of l's support to x's support is over minimum confidence. This idea can be improved in a depth-first fashion based on the property of frequent itemsets. Because the support of any subset x' of x must be as great as the support of x, the confidence of the rule $x' \rightarrow (l - x')$ cannot be more than the confidence of $x \rightarrow (l - x)$. Therefore, if a rule $(l - x) \rightarrow x$ holds, all rules of the form $(l - x') \rightarrow x'$ must also hold. By using this idea, a fast algorithm first generates all rules with frequent single item. Then, all pairs in a rule are generated by using the consequence of those single item rules, and so on, till all rules are found.

4.6.2 Class Diagram

Figure 4.10 shows the UML class diagram for the implementation of the AM. The AssociationMiner (See Appendix A.3.1) class launches a thread to collect visiting

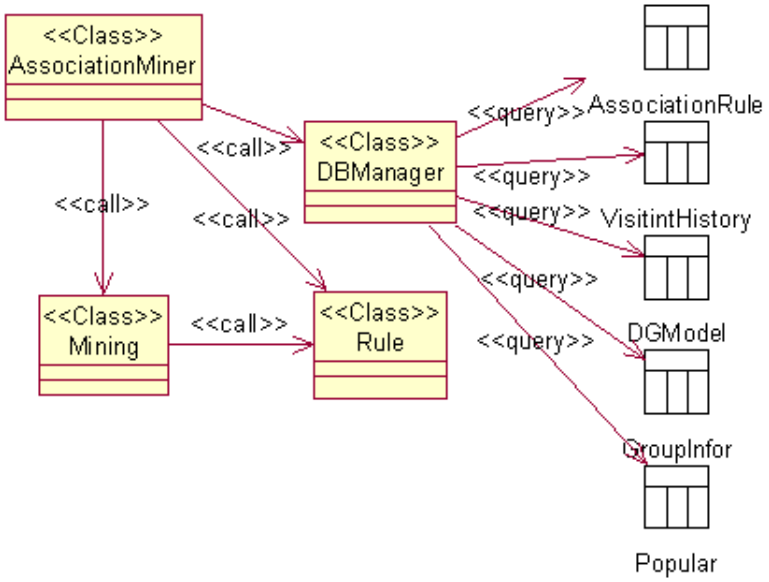


Figure 4.10: UML Class Diagram for the AM

history of and demographic information about users in each group after a period of time. It then calls functions in the Mining class to mine association rules from the history and user property values. The Mining class implements Apriori algorithm to provide the association rule mining function. It also finds out the most popularly visited pages for each user group. The DBManager class contains functions of querying the database. Those functions are called by the AssociationMiner class to retrieve users' visiting history and property values, to retrieve user group information, and to update association rules and information about popularly visited pages.

4.7 Inference Engine

The Inference Engine (IE) is in charge of inferring user's property values according to the information sources and their reliability. The information sources include direct information collected from users through the UPM, user groups information generated by the UGH, association rules discovered by the AM, and general facts specified by the author through the AT.

4.7.1 Information Sources

Users interact with the Profile Editor and may directly provide demographic information about themselves through a registration processes. Such information is defined as direct information.

The UGH generates user groups based on users' visiting history. Users with similar interests are grouped together. We assume that users who have similar interests may have properties in common. The most common property values are found for each group. If some properties of users in a group are missing, they can be decided by the most common property values for this group. Such an information source is so called groups information.

One type of association rule indicates that given a particular user property value, which other user property values also exist. Such an information source can also be used for inferring users' missing property values. Users' missing property values will be inferred according to these association rules and users' current property values which are either directly provided or inferred from groups information.

In the initialization stage, authors define user model description through the AT. They may specify default values for user properties as well. Each pair of property and its default value is called a 'general fact'. This is the traditional way to assign default users' property values.

Reliability of the information sources is defined as follows:

$$Direct > Groups > Association\ \ Rule > General\ \ Facts$$

Direct information has the highest reliability because it is directly provided by users. We assume users would provide reliable information about themselves for the purpose of obtaining relevant response information from adaptive Web systems. Groups information is more reliable than the information of association rules because groups information is generated within the scope of groups, whereas association rules are

discovered within the scope of all existing users who communicate with the adaptive Web systems. The information of general facts has the lowest reliability because they are based on the statistics over property values of users who communicate with not only the clients of GUMSAWS, but also other adaptive Web systems. Inference results from less reliable information sources can be overridden by the ones from more reliable information sources.

4.7.2 Inference Process

For a new user, property values are initialized as default. Property values would be updated if this user provides direct information through the Profile Editor. To infer this user's property values, the IE first checks whether there are user properties whose values are not from direct information source. If the IE finds that there are such user properties, it would infer those user properties from the groups information according to which group this user belongs to, which property values are the most common, and whether support values of the property values are above a threshold. Before inferring user property values from association rules, the IE needs to check again whether there are user properties whose values are determined by the general facts. The property values that are the most relative to the user's current property values (except default ones) will be assigned to the user.

4.8 Recommender

The Recommender is used to recommend pages strongly related to the pages that users are currently reading. Recommendations are provided based on association rules discovered by the AM. The association rules are mined from visiting histories of the users in the same group. The user's visiting history is made up of visited pages in all previous sessions. The association rule here indicates that given a particular page read by users in a group, which other pages the users have also read. The extracted

association rules are ranked based their support and confidence values. Moreover, some popular pages might also be recommended if the number of qualified pages for recommendation is not enough. The service provided by the Recommender is presented in the Table 4.6.

Table 4.6: The Service Provided by the Recommender

Instruction	reqID	Parameters	Description
recommend	10	userID and visitingPage	recommend pages strongly related to the current reading page

4.8.1 Class Diagram

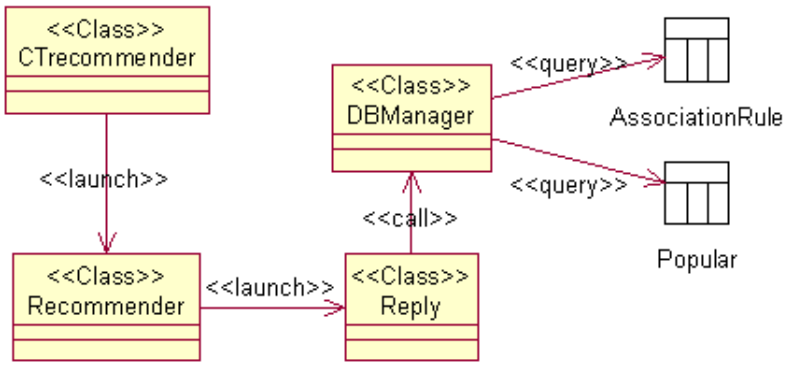


Figure 4.11: UML Class Diagram for the Recommender

The UML class diagram for the implementation of the Recommender is shown in Figure 4.11. The CTrecommender class (See Appendix A.4.1) registers the Recommender to the MI first, and then launches a Recommender thread for accepting requests from the MI. The CTrecommender class also defines the global variables for the database connection, and the IP address and port number of the computer that runs the MI, and specifies the service port for the Recommender (7777). The Recommender class (See Appendix A.4.2) accepts requests from the MI and launches a new Reply thread to handle each request. The Reply class (See Appendix A.4.3)

interprets the request passed from the Recommender class and reads the request ID attached in the request. The request ID will determine which service should be used for handling the request. The DBManager class contains functions of querying the database. These functions are called by the Reply class to handle each request.

4.9 Profile Editor

Figure 4.12: Example of the Profile Editor

The Profile Editor (PE) is implemented as an interface to allow users to see information held about them, and to modify their information. Requests from users received through the PE will be handled directly by the UPM as direct information about users. The PE makes adaptive Web systems transparent in that users have

full control on their information. An example of the PE for adaptive Web systems in E-news domain is shown in Figure 4.12. Users can view their information by loading their profiles, and modify their demographic information, such as age, gender, and so on.

The PE is implemented using J2EE technology. A servlet page provides the form to allow users to load their profiles, and modify profile fields. Modifications made on user profiles will be stored into the database.

4.10 Concluding Remarks

GUMSAWS is implemented by using the java programming language, MySQL database server, and RDF. The implemented system components include the UMD, the GMD, the AT, the MI, the UPM, the UGH, the AM, the IE, the Recommender, and the PE. The structure of implemented classes in some of these system components are presented. The implemented machine learning and data mining techniques, and other algorithms are also described.

The UMD consists of the domain-dependent UMV and the description of application-dependent user model with the default user profile. The GMD consists of the domain-dependent GMV and the description of application-dependent group model with the default group profile. The AT provides the author with the GUI to define user and group models and specify default user and group profiles. The MI forwards adaptive systems' requests to appropriate instructions provided by the components of GUMSAWS, and forwards responses back to adaptive systems. The UPM is responsible for providing adaptive Web systems with information about users and their navigation patterns. The UGH is responsible for grouping users with common interests. The AM is responsible for mining association rules from either users' demographic information or navigation history. The IE is in charge of inferring user's property values according to the information sources (direct information, groups information,

association rules, and general facts) and the reliability of them. The Recommender is used to recommend pages strongly related to the pages that users are currently reading. The PE is implemented as an interface to allow users to see information held about them, and to modify their information.

Chapter 5

Experiments and Results

GUMSAWS has been examined in the E-tailer domain to evaluate the performance of its user modeling task of inferring user property values. The dataset for training and testing is extracted from KDDCUP2000 data. Information sources about the dataset, including direct information, general facts, groups information, and association rules, are presented. The results of different experiments are also described in this chapter.

We carried out different experiments for different purposes. Experiments are carried out to evaluate the accuracy of inferring user property values from different information resources. The average accuracy from the experiments was found to be 67.7%. Experiments are also carried out to compare the performance of inference by using two different clustering algorithms, K-means and K-means+, to generate groups information. The results indicate that the K-means+ algorithm performs slightly better than the K-means algorithm. The performance of inference from different combinations of information sources is compared as well. The results show that the combination of direct information, groups information, association rules, and general facts provides the best accuracy. The inference performance produced by the combination of direct information, groups information and general facts is better than the combination of only the direct information and the general facts.

5.1 Data Set

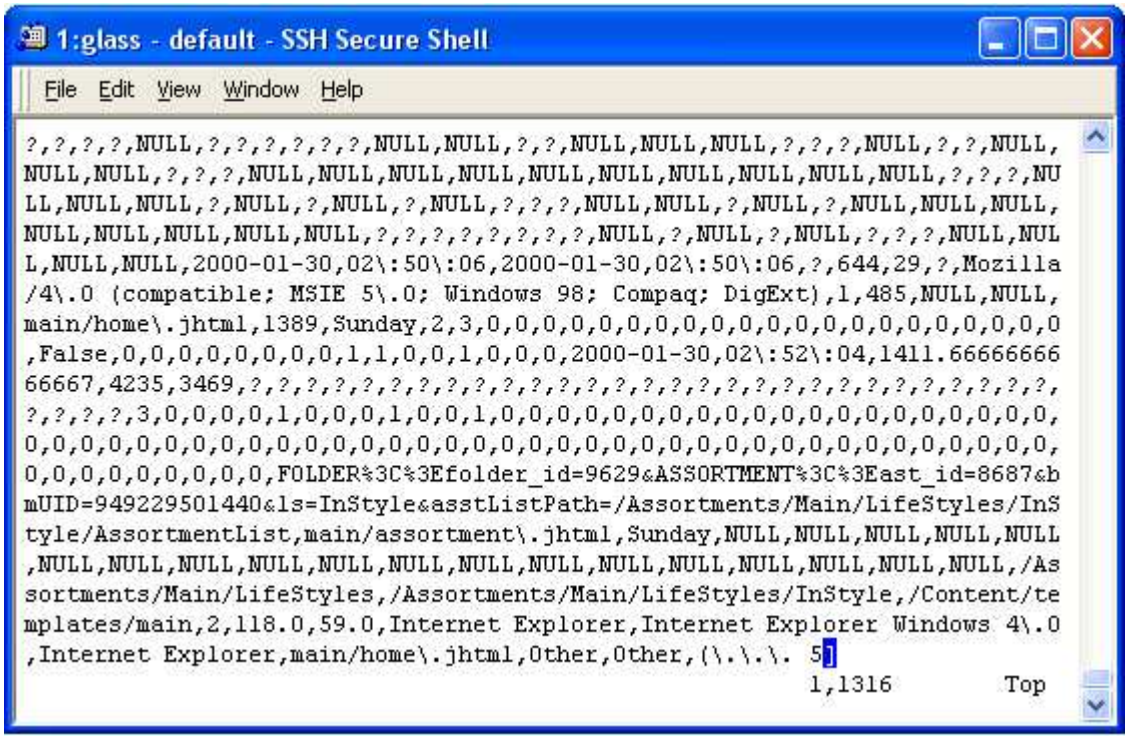


Figure 5.1: One User Session in the Original Dataset

The dataset used for diverse experimentation is the aggregated KDDCUP2000 dataset with the size of 296.9 Megabytes. This dataset contains clickstream and purchase data from Gazelle.com, a legwear and legcare web retailer that closed their online store on 8/18/2000. Information about 234954 user sessions and values of 296 properties for each user and each session is described in the dataset. Figure 5.1 shows information about one of the user sessions and the user who performed this session. Records are separated by the ',' sign. Each record represents the value of the corresponding property. For example, the first record in the figure represents the value of the property of 'Which Do You Wear Most Frequently'. The property values represented by the '?' mark or 'NULL' indicate that no information has been collected about these properties. It happens because not all users have registered at

Gazelle.com. Even for these registered users, they might provide information for only selected properties because of privacy concerns. For example, 3527 users provided information about whether they are working women, but only 294 users provided information about the year of birth.

Table 5.1: Information about Selected Properties

Property	Possible Values	# of Users	Most Common Value
purchase	once a year, each week, every 6 months	2459	every 6 months
marital	Inferred Married, Single, Inferred Single, Married	2974	Married
working	True, False	3527	False
gender	Female, Male	2423	Female

5.1.1 General Facts

We select those properties which have values provided by the large number of users. The selected four properties are 'purchase frequency' ('purchase' in Tables 5.1, 5.5, and 5.6), 'marital status' ('marital' in Tables 5.1, 5.5, and 5.6), 'working women or not' ('working' in Tables 5.1, 5.5, and 5.6) , and 'gender' information. Table 5.1 presents information about the possible values of these properties, total number of users who provided information about each property, and the most common value for each property. The most common values and their corresponding properties represent general facts. For example, one of the general facts indicated in the table is that 'Female' is the most common value of the property 'gender'. General facts are mined from the whole dataset.

5.1.2 Preprocessed Dataset

The original dataset is preprocessed by extracting users who have registered and provided information about the four properties. After preprocessing, 1246 users are

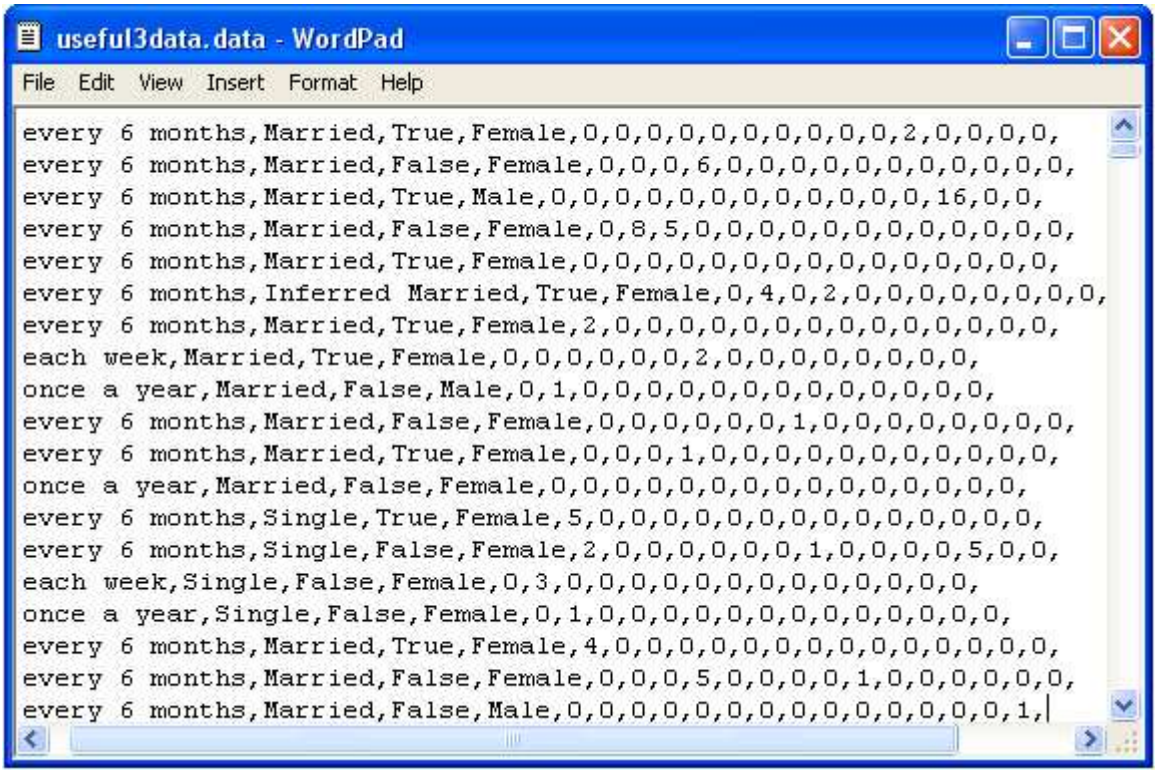


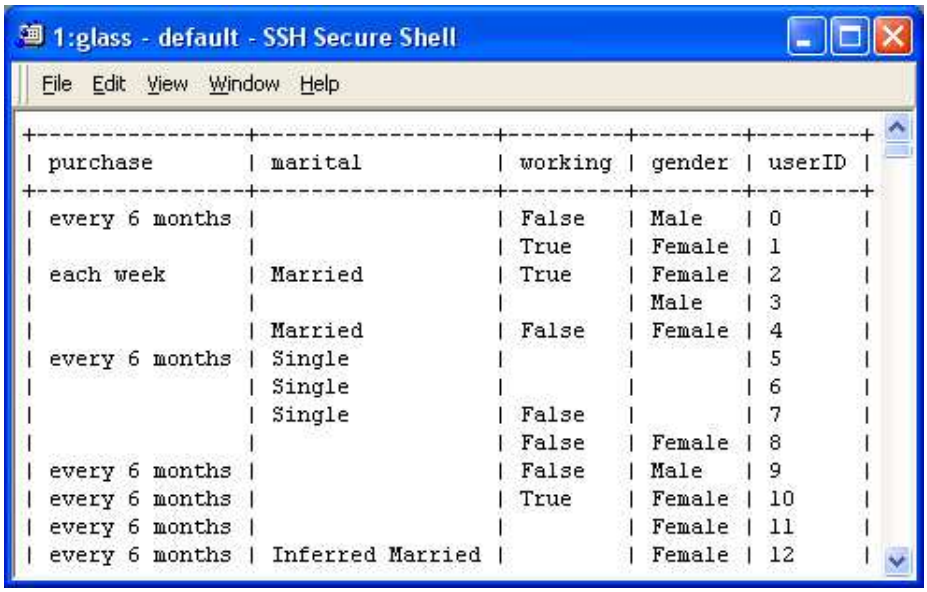
Figure 5.2: Information about the Processed Dataset

chosen to be involved in the evaluation. The values of those four properties and the information about users' navigation history are also extracted. The snapshot of the preprocessed dataset is shown in Figure 5.2. It contains values of these four user properties followed by the number of visited pages in each product category. Values are separated by the ',' .

5.2 Training Data

1246 users and four user properties are selected and used in the evaluation. For the purpose of testing the accuracy of inferring user property values, we randomly set aside 10% of property values for each property. We repeat this process 10 times to acquire 10 data sets listed in the table 5.5. As shown in Figure 5.3, some of the users' property values (for example, the value of the property 'purchase' for the user '1')

are missing. The rest is for training. From the training data, groups were generated by the UGH, and association rules were discovered by the AM. The IE will infer the missing property values from groups information, association rules, and general facts mined earlier. Information about user groups and discovered association rules are described in this section.



purchase	marital	working	gender	userID
every 6 months		False	Male	0
		True	Female	1
each week	Married	True	Female	2
			Male	3
	Married	False	Female	4
every 6 months	Single			5
	Single			6
	Single	False		7
		False	Female	8
every 6 months		False	Male	9
every 6 months		True	Female	10
every 6 months			Female	11
every 6 months	Inferred Married		Female	12

Figure 5.3: Missing Property Values

5.2.1 Groups

The Usage Group Handler (UGH) generates user groups by using clustering algorithms. Two clustering algorithms, K-means and K-means+, are implemented and integrated into the UGH. The UGH groups users together according to their interests. The user's interest is represented by the number of pages read by this user in the fifteen product categories, such as TH (thigh-highs) category, WDCS (women's dress/casual socks) category, PH (pantyhose) category, and so on. Examples of groups are represented in Tables 5.2 and 5.3. Five groups (indicated by their IDs) are generated by both of the K-means and K-means+ algorithms. However, generated groups

by using those two algorithms are different in terms of the distribution of users and the group centers. The distribution of users is represented by the population of members in each group (P in Tables 5.2 and 5.3). Group center is a normalized vector of average interest value of all group members.

Table 5.2: User Groups Generated by Using the K-means Algorithm

ID	P	PH	WDCS	TH	WAS	FO	LG	TT
1	124	0.002	0.003	0.000	0.000	0.000	0.090	0.014
2	370	0.001	0.018	0.000	0.004	0.000	0.001	0.002
3	241	0.005	0.001	0.001	0.000	0.001	0.019	0.998
4	442	0.000	0.000	0.000	0.001	0.000	0.005	0.012
5	69	0.001	0.003	0.000	0.000	0.001	0.079	0.009
ID	MDS	MAS	BKS	LEO	GDCS	KP	BDCS	MCS
1	0.014	0.005	0.000	0.000	0.002	0.996	0.000	0.004
2	0.990	0.126	0.000	0.000	0.000	0.017	0.000	0.064
3	0.013	0.010	0.000	0.041	0.000	0.034	0.000	0.003
4	0.029	0.998	0.000	0.004	0.000	0.020	0.000	0.051
5	0.011	0.000	0.052	0.798	0.078	0.000	0.078	0.585

Table 5.3: User Groups Generated by Using the K-means+ Algorithm

ID	P	PH	WDCS	TH	WAS	FO	LG	TT
1	139	0.921	0.052	0.017	0.010	0.010	0.013	0.052
2	607	0.008	0.008	0.045	0.004	0.039	0.014	0.003
3	82	0.0207	0.051	0.017	0.017	0.000	0.002	0.926
4	95	0.022	0.194	0.007	0.875	0.002	0.000	0.020
5	323	0.020	0.948	0.010	0.029	0.006	0.008	0.006
ID	MDS	MAS	BKS	LEO	GDCS	KP	BDCS	MCS
1	0.008	0.005	0.000	0.004	0.096	0.064	0.000	0.005
2	0.060	0.117	0.003	0.049	0.005	0.106	0.005	0.041
3	0.014	0.019	0.000	0.044	0.000	0.031	0.000	0.003
4	0.017	0.135	0.000	0.000	0.000	0.016	0.000	0.003
5	0.018	0.021	0.000	0.003	0.000	0.049	0.000	0.012

5.2.2 Association Rules

The information source of association rules is generated by the Association Miner from all user property values. The support and confidence values of the association rules must be above a threshold to be considered. Therefore, the number of discovered association rules would be different if the setting for the thresholds (minimal support and minimal confidence) was different. Table 5.4 presents the results of the number of discovered association rules with the corresponding setting for the minimal support and minimal confidence (M_S and M_C in Table 5.4).

Table 5.4: Number of Discovered Association Rules

# of Rules	M_S	M_C
0	0.8	0.8
5	0.6	0.6
13	0.5	0.5
21	0.4	0.4
31	0.3	0.3
47	0.2	0.2
55	0.1	0.1

Some examples of association rules with confidence and support values are presented as follows:

[working:False] → [purchase:every 6 months] (0.823, 0.51)

[gender:Female] → [working:False] (0.598, 0.55)

...

[working:False, purchase:every 6 months] → [gender:Female] (0.898, 0.458)

[purchase:every 6 months, gender:Female] → [working:False] (0.608, 0.458)

...

[working:False, marital:Married, purchase:every 6 months]

→ [gender:Female] (0.939, 0.337)

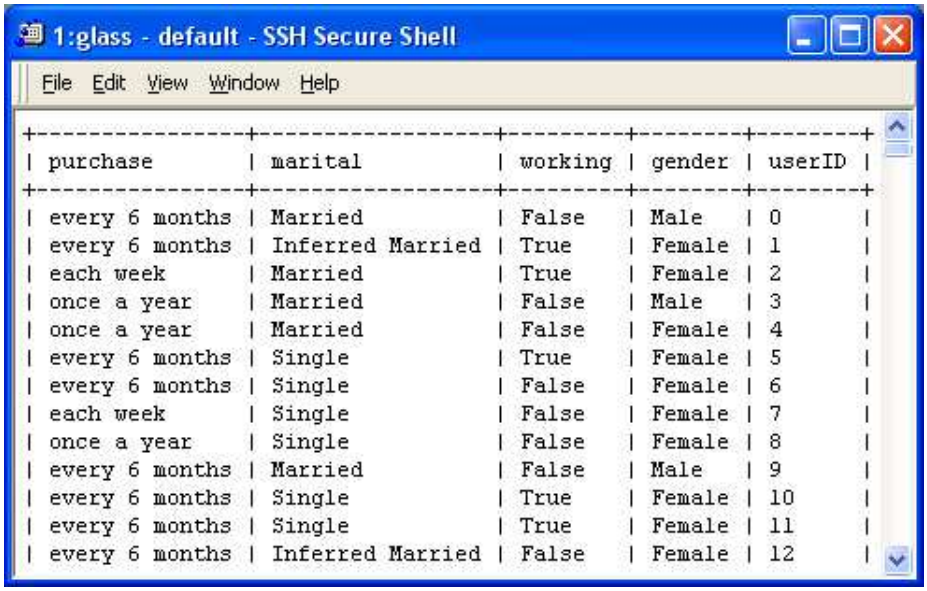
[working:False, purchase:every 6 months, gender:Female]

→ [marital:Married] (0.736, 0.337)

The first association rule presented in the examples is that 'working' property value of 'False' implies 'purchase' property value of 'every 6 months' with confidence value of 0.823 and support value of 0.51. This association rule indicates that users who are not working women normally purchase once every six months.

5.3 Experimental Results

Based on the training and testing data, experiments are carried out to evaluate the performance of inferring user property values from information sources, to compare the performance of inference by using the K-means and K-means+ algorithms to generate groups information, and to compare the performance of inference from different combinations of information sources. This section presents results from those experiments.



purchase	marital	working	gender	userID
every 6 months	Married	False	Male	0
every 6 months	Inferred Married	True	Female	1
each week	Married	True	Female	2
once a year	Married	False	Male	3
once a year	Married	False	Female	4
every 6 months	Single	True	Female	5
every 6 months	Single	False	Female	6
each week	Single	False	Female	7
once a year	Single	False	Female	8
every 6 months	Married	False	Male	9
every 6 months	Single	True	Female	10
every 6 months	Single	True	Female	11
every 6 months	Inferred Married	False	Female	12

Figure 5.4: Inferred Property Values

Table 5.5: Accuracy of Inferring User Property Values

Test #	purchase	marital	working	gender	Accuracy
1	0.736	0.616	0.544	0.840	0.684
2	0.760	0.600	0.536	0.824	0.680
3	0.688	0.592	0.576	0.864	0.680
4	0.704	0.592	0.584	0.784	0.666
5	0.664	0.600	0.560	0.792	0.654
6	0.760	0.576	0.512	0.856	0.676
7	0.728	0.584	0.600	0.864	0.694
8	0.760	0.648	0.480	0.800	0.672
9	0.696	0.616	0.592	0.864	0.692
10	0.664	0.600	0.512	0.824	0.650
Average					0.676

5.3.1 Inference Accuracy

As shown in Figure 5.3, some of the users' property values are missing. After the inference process, these users' missing property values are inferred based on the information sources of direct information, groups information, association rules, and general facts. Inferred property values are shown in Figure 5.4. For example, the value of property 'purchase' of the user whose 'userID' is '1' is inferred as 'every 6 months'. According to Figure 5.3, there are other users whose values of property 'purchase' are inferred. Their 'userID' values are '3', '4', '6', '7' and '8'. The values of property 'marital' for the users whose 'userID' values are '0', '1', '3', '8', '9', '10' and '11' are inferred. The values of property 'working' for the users whose 'userID' values are '3', '5', '6', '11' and '12' are inferred. The values of property 'gender' for the users whose 'userID' values are '5', '6' and '7' are inferred.

Inference accuracy is calculated as the average ratio of the number of correctly inferred values for each of the four properties to the number of missing values of this property. Results of accuracy are presented in Table 5.5. The average accuracy is 67.6%, which is calculated after setting aside the highest and lowest values.

Table 5.6: Comparison of K-means and K-means+

Test #	Algorithm	purchase	marital	working	gender	Average
1	K-means	0.800	0.584	0.448	0.896	0.682
	K-means+	0.800	0.608	0.448	0.896	0.688
2	K-means	0.776	0.608	0.560	0.896	0.71
	K-means+	0.776	0.704	0.560	0.896	0.734
3	K-means	0.728	0.544	0.52	0.88	0.668
	K-means+	0.76	0.624	0.52	0.88	0.696
4	K-means	0.752	0.552	0.448	0.92	0.668
	K-means+	0.752	0.632	0.448	0.92	0.688
5	K-means	0.752	0.632	0.512	0.872	0.692
	K-means+	0.752	0.688	0.512	0.872	0.706

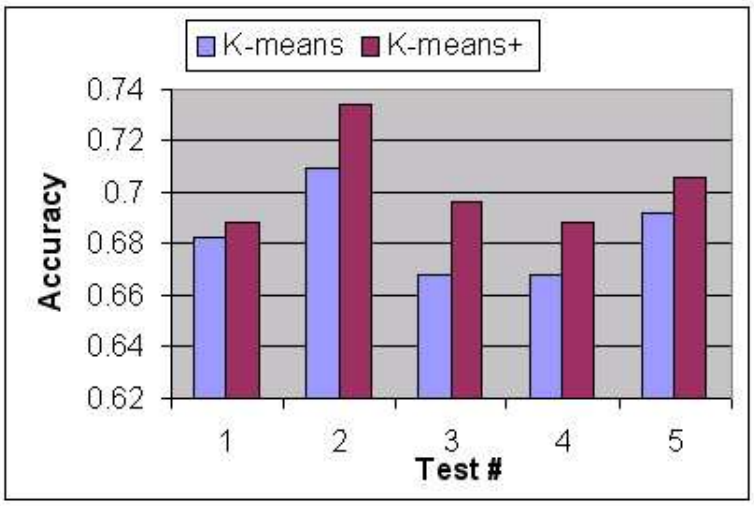


Figure 5.5: Comparison of K-means and K-means+

5.3.2 Comparison of K-means and K-means+

Both the K-means and K-means+ algorithms are used to generate groups information. Experiments are carried out to compare the inference performance from the information source of groups information generated by using the two algorithms. Results presented in Table 5.6 and the Figure 5.5 show that accuracy of property value prediction performed by the K-means+ algorithm is slight better than that of the K-means algorithm.

The improvement is due to the fact that the K-means+ algorithm splits clusters

with outliers and uses the multi-centered clusters to obtain better performances. The K-means+ algorithm has a splitting procedure that removes outliers from existing clusters to form new clusters. An outlier is an object that is far from the majority of the objects in a cluster. As shown in Figure 5.6, the center of the cluster X is represented by the object c. The distance d between c and one of the members of the cluster, p, is greater than a threshold. The object p is deemed an outlier. The outliers of the cluster X are formed as a news cluster Y, as shown in Figure 5.7. The K-means+ algorithm also has a linking procedure. Some adjacent clusters may be linked to form a larger cluster if they are close enough. The centers of linked clusters are intact after linking; therefore, the newly formed clusters are multi-centered, and they can be in arbitrary shapes, such as a spatial chain.

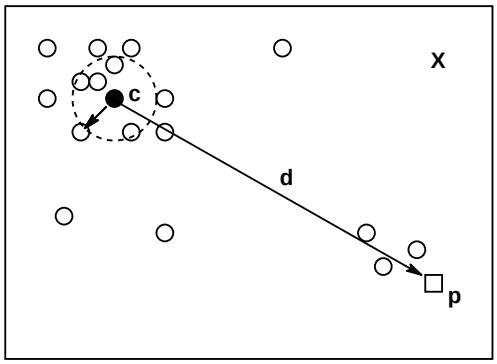


Figure 5.6: Clusters before Splitting

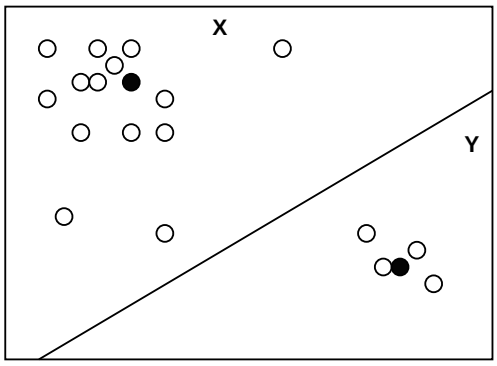


Figure 5.7: Clusters after Splitting

Table 5.7: Comparison of Different Combinations of Information Sources

Test	DI&GF	DI&GI&GF	DI&GI&AR&GF
1	0.622	0.658	0.680
2	0.570	0.632	0.666
3	0.594	0.622	0.654
4	0.616	0.662	0.676
5	0.618	0.664	0.694
Average	0.604	0.648	0.674

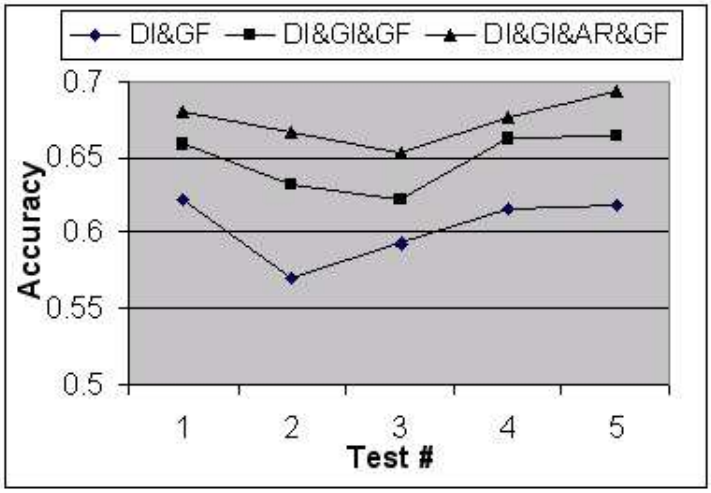


Figure 5.8: Comparison of Different Combinations of Information Sources

5.3.3 Comparison of Different Combinations of Information Sources

Users' missing property values are inferred from the information sources of direct information, groups information, association rules, and general facts. We carried out experiments to compare the inference performance from different combinations of information sources. For the later user, three notions are defined as follows:

- **DI&GF**: the combination of the information sources, direct information and general facts;
- **DI&GI&GF**: the combination of the information sources, direct information, groups information and general facts;

- **DI&GI&AR&GF**: the combination of the information sources, direct information, groups information, association rules, and general facts.

The experimental results are presented in Table 5.7 and Figure 5.8. The results indicate that the combination of direct information, groups information, association rules, and general facts provides the best performance. The inference performance produced by the combination of direct information, groups information and general facts is better than by the combination of only the direct information and the general facts.

5.4 Concluding Remarks

KDDCUP2000 data is used to evaluate the performance of inferring user property values. The dataset is preprocessed. 1246 users and four user properties are selected and used in evaluation. Information sources of general facts, groups information, and association rules are generated from the dataset. Experiments are carried out to evaluate the performance of inferring user property values from information sources, to compare the performance of inference by using the K-means algorithm and K-means+ algorithm to generate groups information, and to compare the performance of inference from different combinations of information sources.

Conclusions are drawn from the experiments. The average accuracy accuracy of inferring user property values from information resources is 67.7%. This result can be improved by introducing more information sources, such as the information source of subgroups. Subgroups can be generated by re-clustering of users in the same group. The K-means+ algorithm performs slightly better than the K-means algorithm. The combination of direct information, groups information, association rules, and general facts provides the best accuracy of inference. The inference performance produced by the combination of direct information, groups information and general facts is better than the combination of only the direct information and the general facts.

Chapter 6

Example of GUMSAWS in Use

The previous chapter examined GUMSAWS in the E-tailer domain by evaluating its performance of inferring user property values from diverse information sources collected or mined by the system. This chapter describes an example of adaptive Web system that illustrates basic user modeling functions offered by GUMSAWS and its user modeling task of providing recommendations based on user navigation history.

6.1 PENS

Personalized Electronic News System (PENS) is an adaptive Web news system. PENS is implemented based on the adaptive Web construction framework. It is developed by the Intelligent and Adaptive Systems (IAS) group at the Faculty of Computer Science, University of New Brunswick. PENS presents news to the users taking advantage of context information such as location and behavior. It also adapts the presentation of the news based on device characteristics, such as screen size and color capabilities. PENS is implemented as the proof of concept, to show adaptation based on users' behavior and client-side characteristics, and to demonstrate the use of GUMSAWS. The use of GUMSAWS here is to provide data sources that shape the dynamic aspect of PENS. The data provided by GUMSAWS about users is used to make decisions

for adaptation and for populating the under-construction page.

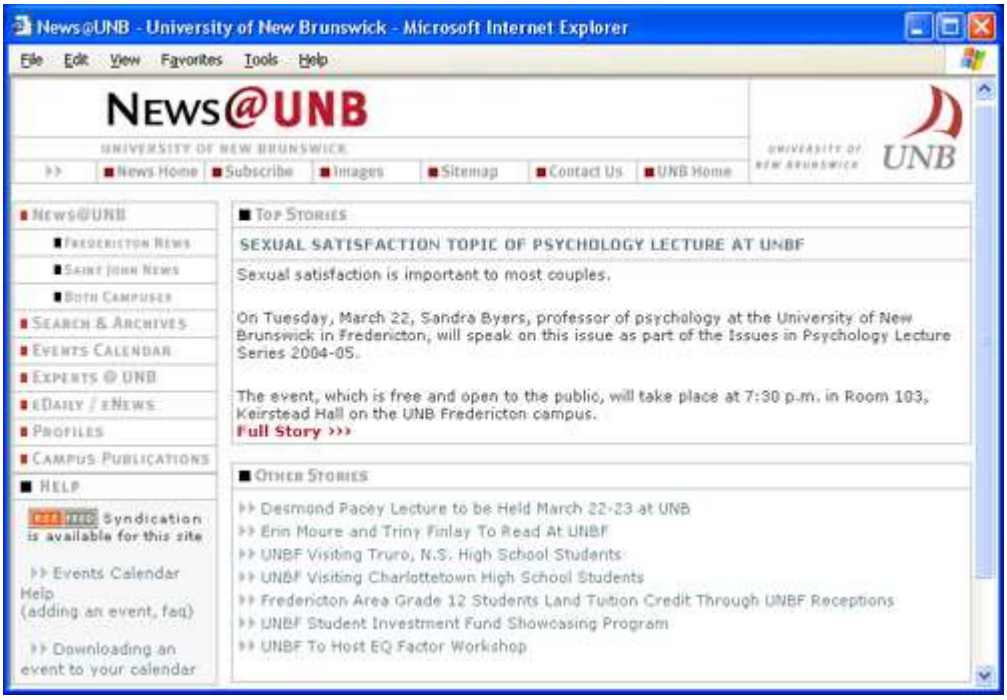


Figure 6.1: The NEWS@UNB Website

PENS partially imitates the NEWS@UNB website (shown in Figure 6.1), which news items are gathered from. News items are acquired from the news feed in the Rich Site Summary (RSS) format provided by the NEWS@UNB website. The news feed is written in XML, and the news items in it are retrieved through the JDOM API. Although the original NEWS@UNB website is static and does not provide any adaptation to users, PENS provides three types of adaptation, content adaptation, navigation adaptation, and presentation adaptation.

6.2 Web Pages

Three different types of Web pages, the front page, the category based news page and the full news page, are generated by PENS. As shown in Figure 6.2, the front page has two parts. The left part lists news categories in the "NEWS SECTIONS"

Figure 6.2: The Front Page

section. Clicking one of the category names in this section will lead the user to the category based news page (shown in Figure 6.3) which shows the recent news items in this category. The right part lists three most recent news items with their titles and first few statements in the "TOP NEWS" section and four other news items with only titles in the "MORE TOPICS" section. Clicking the title of the news will lead the user to the full news page. The Figure 6.3 shows the category based news page for the academic category. Three recent news items in this category are listed in this page. There is a link of the bottom of the page that allows the user to return to the front page. As shown in Figure 6.4, the full news page presents full information of a

Figure 6.3: The Category Based Page

news item, including its topic, publication date, correspondent, and news body. The news banner is located at top of all pages, the front page, the category based page, and the full news page.

6.3 Adaptation

Users may see different pages because of the adaptation provided by PENS. GUMSAWS keeps track of user navigation history which is composed of news items that

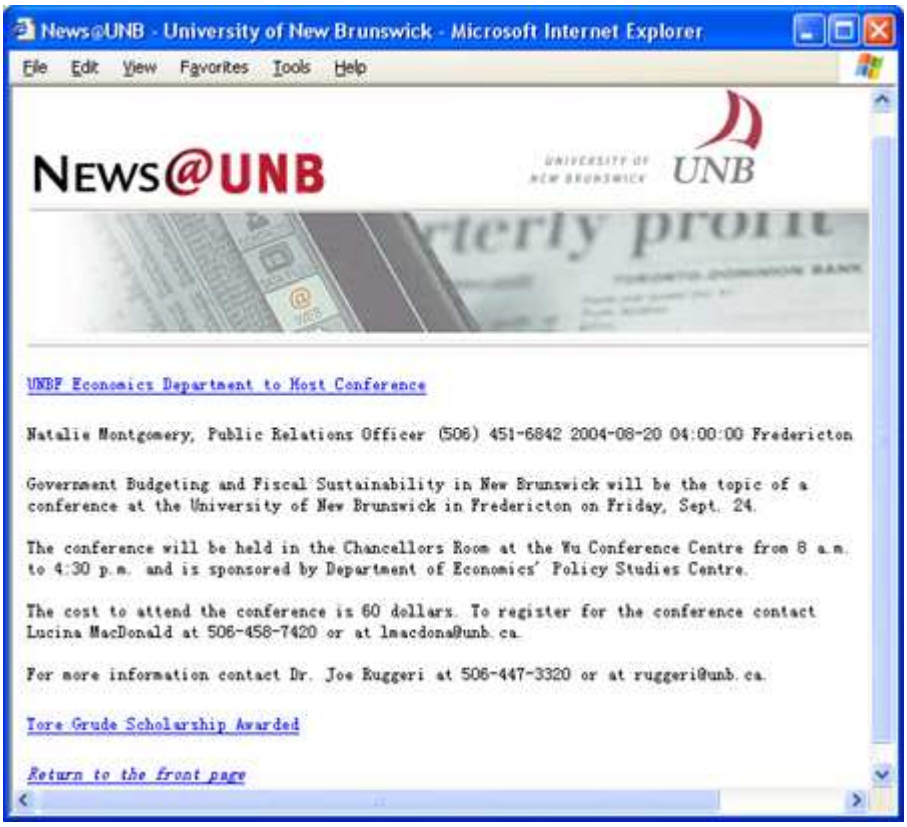


Figure 6.4: The Full News Page

the user has read. User navigation history also indicates how much the user is interested in each news category. PENS provides three types of adaptation based on user navigation history.

In the "NEWS SECTIONS" section on the front page, news categories are sorted and listed based on user interests in different categories. As shown in Figure 6.2, the category "ACADEMIC" is on the top of the link list, which indicates that the user is more interested in the academic category than other categories. User's interest in each category is determined by the number of news items in this category that have been read by the user. Moreover, the earlier the news item has been read, the less weight it will have in the category that the news item belongs to because the user's interest might change over time.

In the "TOP NEWS" section on the front page, the first few statements of the

news item that the user has read will not be shown any more. This adaptation is based on the assumption that users would not be interested in reading the first few statements again if they have already read the full body of the news. The examples are the second news item in the "LATEST NEWS" section as shown in Figure 6.2, and the first news item in the category based page for the academic category shown in Figure 6.3.

Another adaptation is news recommendation, which is an example of navigation adaptation. On the bottom of the full news page shown in Figure 6.4, there is a link to a related news item. This news item is the recommendation provided by the Recommender of GUMSAWS. The Recommender is used to recommend news items strongly related to the news stories that users are currently reading. Recommendations are provided based on association rules discovered by the AM. The association rules are mined from visiting histories of the users in the same group. The extracted rules are ranked based on their support and confidence values. Moreover, some popular news items or the latest news items might also be recommended if the number of qualified news items for recommendation is not enough.

6.4 Concluding Remarks

PENS is implemented to demonstrate the user of GUMSAWS. The use of GUMSAWS here is to provide data sources that shape the dynamic aspect of PENS. PENS partially imitates the NEWS@UNB website. It generates three different types of Web pages, the front page, the category based news page, and the full news page. Three types of adaptation are provided by PENS based on user navigation history kept track of by GUMSAWS. User navigation history indicates how much the user is interested in each news category. News catgories are sorted and listed based on user interests in different categories. The first few statements of the news item that the user has read will not be shown any more. The third adaptation is news recommendation.

Chapter 7

Conclusions and Future Work

7.1 Literature Review

The user modeling area has been explored to clarify the terms of user profile, user model, and user modeling. A user profile is defined as a collection of information about a user, and is an instance of user model for a particular user. A user model is an abstract representation which contains explicit assumptions on all aspects of the user that may be relevant for the behaviour of the system. User modeling is the whole process of constructing user models and creating, updating or deleting user profiles. It consists of two processes, collecting data about users and processing the data to build or update the user model. We summarized the methods of collecting data about users and their problems. We also introduced machine learning and data mining techniques that are used to build user models, and some example systems, such as the Lumiere project, Syskill & Webert system, Web-EasyMath, and SeAN.

The user modeling history from user modeling components to user modeling shell systems and servers has been introduced. User modeling components are embedded in application systems, and lack of reusability. User modeling shell systems form an "empty" user modeling mechanism. They become part of the application after being filled with application-dependent user modeling knowledge. They receive information

about the user from the application only and supply the application with assumptions about the user. User modeling servers are centralized user modeling components for more than one applications in a similar domain. They assist these applications concurrently.

Some examples of generic user modeling shell systems and servers have been described. A few issues, such as the representation of user model, the maintenance of user model, and the acquisition of user model, have been mainly discussed to show generality, expressiveness, and strong inferential capability of these shell systems and servers. We also discussed the advantages and disadvantages of them.

7.2 Design and Implementation of GUMSAWS

GUMSAWS has been designed and implemented to provide basic user modeling functions, to have the capabilities of domain-independent user modeling, and to communicate with the adaptive Web systems through a network. The design of GUMSAWS reaches the goal of generality, extendability, and replaceability. Five major subsystems, the Model Maintainer, the Information Source Generator, the Recommendation Provider, the System Repository, and the Model Description, have been designed as the groups of system components.

GUMSAWS has been implemented by using Java, MySQL database server, and the RDF. The implemented system components include the User Model Description (UMD), the Group Model Description (GMD), the Authoring Tool (AT), the Modeling Interface (MI), the User Profile Manager (UPM), the Usage Group Handler (UGH), the Association Miner (AM), the Inference Engine (IE), the Recommender, and the Profile Editor (PE). The structure of implemented classes in some of these system components has been presented. The implemented machine learning and data mining techniques, and other algorithms have been also described.

The UMD consists of the domain-dependent UMV and the description of application-dependent user model with the default user profile. The GMD consists of the domain-dependent GMV and the description of application-dependent group model with the default group profile. The AT provides the author with a GUI to define user and group models and specify the default user and group profiles. The MI forwards adaptive systems' requests to appropriate instructions provided by the GUMSAWS components, and forwards responses back to the adaptive systems. The UPM is responsible for providing adaptive Web systems with information about users and their navigation patterns. The UGH is responsible for grouping users with common interests. The AM is responsible for mining association rules from either users' demographic information or user navigation history. The IE is in charge of inferring users' property values according to the information sources (direct information, groups information, association rules, and general facts) and the reliability of them. The Recommender is used for recommending pages strongly related to the pages that users are currently reading. The PE is implemented as an interface to allow users to see information held about them, and to modify their information.

The implemented GUMSAWS acts as a centralized user modeling component and is able to assist many adaptive Web systems concurrently. It supplies the adaptive systems with information about users. The data provided by GUMSAWS is used by these adaptive systems to make decisions for different types of adaptation. User models built by GUMSAWS can be shared among all these adaptive systems. In addition, two main user modeling tasks were highlighted in GUMSAWS, inferring user property values and providing recommendation based on user navigation history.

7.3 Evaluation and Demonstration

KDDCUP2000 data has been used to evaluate the performance of inferring user property values. The dataset has been preprocessed. 1246 users and the four user properties have been selected and used in evaluation. Information sources of general facts, groups information, and association rules have been generated from the dataset. Experiments have been carried out to evaluate the performance of inferring user property values from information sources, to compare the performance of inference by using the K-means algorithm and K-means+ algorithm to generate groups information, and to compare the performance of inference from different combinations of information sources.

Conclusions have been drawn from the experiments. The average accuracy of inferring user property values from different information resources is 67.7%. The K-means+ algorithm performs slightly better than the K-means algorithm. The combination of direct information, groups information, association rules, and general facts provides the best accuracy of inference. The inference performance produced by the combination of direct information, groups information and general facts is better than the combination of only the direct information and the general facts.

An example application, PENS, has been implemented to illustrate how the data sources provided by GUMSAWS can be used for adaptation. Three different types of Web pages, the front page, the category based news page and the full news page, are generated by PENS. PENS also provides three types of adaptation, content adaptation, navigation adaptation, and presentation adaptation.

7.4 Future Work

There are many opportunities that the research and implementation can be extended in the near future to improve GUMSAWS for real applications.

- Authors define application-dependent user and group models and specify the default user and group profiles based on the predefined user and group model vocabularies. However, the user and group model vocabularies might not be complete enough to cover all user and group concepts and relationships in the specific domain. Authors should be allowed to define user and group concepts and relationships that have not existed in the predefined vocabularies.

- Current user modeling systems build user models without or only partially considering the domain models of adaptive systems. However, the user model definition can not be totally separated from the domain model. For example, the overlay user model keeps every concept and associated attribute in the domain model of the adaptive system. It would be helpful if user modeling systems can import the domain models of the adaptive systems and allow authors define user models from the concepts and associated attributes in their domain models.

- The PE, a generic tool, is implemented to make adaptive Web systems transparent. However, the current PE can only allow users view and modify their profiles. It should be able to allow users to define which applications are allowed to see each part of the user model, to control the information sources that should be made available to each application, to view inference explanation for each inferred property values, and to define which part of the user model can be used for inference processes.

- Currently, the communication between GUMSAWS and its clients is through UPD protocol, which is not secure. Other possible secure protocols can be implemented in the system, such as SSL.

Bibliography

[1] F. Abbattista, M. Degemmis, O. Licchilli, P. Lops, G. Semeraro, and F. Zambetta, *Improving the usability of an e-commerce web site through personalization*, Proceedings of the Workshop on Recommendation and Personalization in eCommerce of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems (F. Ricci and B. Smyth, eds.), May 2002, pp. 20–29.

[2] Rakesh Agrawal and Ramakrishnan Srikant, *Fast algorithms for mining association rules*, Proceedings of the 20th International Conference of Very Large Databases (VLDB) (Santiago, Chile), September 1994, pp. 487–499.

[3] Liliana Ardissono, Luca Console, and Ilaria Torre, *An adaptive system for the personalized access to news*, AI Communications **14** (2001), no. 3, 129–147.

[4] D. Billsus and M. Pazzani, *Revising user profiles: The search for interesting web sites*, Proceedings of the Third International Workshop on Multistrategy Learning, AAAI Press, 1996.

[5] D. Billsus and M. J. Pazzani, *A hybrid user model for news story classifcation*, User Modeling: Proceedings of the Seventh International Conference, UM'99 (J. Kay, ed.), Springer, 1999, pp. 99–108.

[6] C. Boyle and A. O. Encarnacion, *Metadoc: An adaptive hypertext reading system*, User Modeling and User-adapted Interaction **4** (1994).

[7] Giorgio Brajnik and Carlo Tasso, *A shell for developing nonmonotonic user modeling systems*, International Journal of Human-Computer Studies **40** (1994), 31–62.

[8] Peter Brusilovsky, *Methods and techniques of adaptive hypermedia*, User Modelling and User-Adapted Interaction **6** (1996), no. 2-3, 87–129.

[9] Brian D. Davison and Haym Hirsh, *Predicting sequences of user actions*, Notes of the AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis (Madison, Wisconsin), 1998.

[10] Paul De Bra, A. Aerts, D. Smits, and N. Stash, *AHA! version 2.0, more adaptation flexibility for authors*, Proceedings of the AACE ELearn'2002 conference, October 2002.

[11] Josef Fink and Alfred Kobsa, *A review and analysis of commercial user modeling servers for personalization on the World Wide Web*, User Modeling and User-Adapted Interaction **10** (2000), no. 3-4, 209–249, Special Issue on Deployed User Modeling.

[12] Josef Fink, Alfred Kobsa, and Andreas Nill, *GUMS: A general user modeling shell*, pp. 411–430, Springer, Berlin, Heidelberg, 1989.

[13] R. Ghani, R. Jones, D. Mladenic, K. Nigam, and S. Slattery, *Data mining on symbolic knowledge extracted from the web*, Proceedings of the Six International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining (Boston, MA), July 2000, pp. 29–36.

[14] I A S group, *Adaptive web sites (AWS) framework high-level design document*, Tech. Report TR03-102, Intelligent and Adaptive Systems Research Group, Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada, December 2003.

[15] Yu Guan, Ali A. Ghorbani, and Nabil Belacel, *Y-means: a clustering method for intrusion detection*, Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE-2003), May 4-7 2003.

[16] ———, *K-means+: An autonomous clustering algorithm*, Submitted to Pattern Recognition (2004).

[17] Eric Horvitz, Jack Breese, David Heckerman, David Hovel, and Koos Rommelse, *The lumiere project: Bayesian user modeling for inferring the goals and needs of software users*, Proceedings of the Fourteenth Conderence on Uncertainly in Artificial intelligentce (Madison, WI), Morgan Kaufmann Publishers, July 1998, pp. 256–265.

[18] Mahesh Joshi and Vipin Kumar, *Tutorial on high performance data mining*, Taught at Fifth International Conference on High Performance Computing (HiPC'98), Chennai, India; and at Joint International Parallel Processing Symposium/Symposium on Parallel and Distributed Processing (IPPS/SPDP)'99, San Juan, Puerto Rico, 1999.

[19] Robert Kass and Tim Finin, *Rules for the implicit acquisition of knowledge about the user*, Proceedings of the AAAI-87 conference, 1987.

[20] ———, *A general user modelling facility*, Proceedings of the SIGCHI conference on Human factors in computing systems, 1988.

[21] Judy Kay, *The um toolkit for cooperative user modelling*, User Modelling and User-Adapted Interaction, Kluwer **4** (1995), no. 3, 149–196.

[22] Judy Kay, Bob Kummerfeld, and Piers Lauder, *Personis: A server for user models*, Proceedings of Adaptive Hypertext 2002, Springer-Verlag, 2002, pp. 203–212.

[23] Mark Kilfoil, Dr. Ali Ghorbani, Wenpu Xing, Zhong Lei, Jing Lu, Jie Zhang, and Xiaowen Xu, *Toward an adaptive web: The state of the art and science*, Proceedings of Communication Network and Services Research (CNSR) 2003 Conference (Moncton, NB, Canada), May 15–16 2003, pp. 108–119.

[24] A. Kobsa, *User modeling: Recent work, prospects and hazards*, M. SchneiderHufschmidt, T. Khme and U. Malinowski (eds.): Adaptive user interfaces: Principles and practice. Amsterdam: North-Holland, 1993.

[25] Alfred Kobsa, *Modeling the user's conceptual knowledge in bgp-ms, a user modeling shell system*, Computational Intelligence **6** (1990), 193–208.

[26] ———, *Supporting user interfaces for all through user modeling*, Proceedings of the Sixth International Conference on Human-Computer Interaction, vol. 1, 1995, pp. 155–157.

[27] ———, *Generic user modeling systems*, User Modeling and User-Adapted Interaction **11** (2001), no. 1-2, 49–63.

[28] Alfred Kobsa, dietmar Muller, and Andreas Nill, *KN-AHS: An adaptive hypertext client of the user modeling system bgp-ms*, Proceedings of the Fourth International Conference on User Modeling, 1994, pp. 99–105.

[29] Alfred Kobsa and Wolfgang Pohln, *The user modeling shell system bgp-ms*, User Modeling and User-Adapted Interaction **4** (1995), no. 2, 59–106.

[30] W. Lam and J. Mostafa, *Modeling user interest shift using a bayesian approach*, Journal of the American Society for Information Science and Technology **52** (2001), no. 5, 416–429.

[31] John Lei and Ali Ghorbani, *The reconstruction of the interleaved sessions from a server log*, Proceedings of the Seventeenth Conference of the Canadian Society for Computational Studies of the Intelligence (Canada), 2004, pp. 133–145.

[32] Isabel Machado, Alexandre Martins, and Ana Paiva, *One for all and all in one - a learner modelling server in a multi-agent platform*, the Seventh International Conference on User Modelling, Judy Kay (ed.), Springer Verlag, 1999, pp. 211–221.

[33] J. B. MacQueen, *Some methods for classification and analysis of multivariate observations*, Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability (Berkeley, Califonia), University of California Press, May 4-7 1967, pp. 281–297.

[34] Frank Manola and Eric Miller, *Rdf primer*, http://www.w3.org/TR/rdf-primer/ (2004).

[35] Mirza B. Murtaza, Jaymeen R. Shah, and Vipul K. Gupta, *A model for designing adaptive e-commerce sites*.

[36] MySQL, http://www.mysql.com, *Mysql database server*.

[37] Mehran Nadjarbashi-Noghani and Ali Ghorbani, *Improving the referrer-based web log session reconstruction*, Proceedings of Communication Network and Services Research (CNSR) 2004 Conference (Fredericton, NB, Canada), May 19–21 2004, pp. 286–292.

[38] Un Yong Nahm and Raymond J. Mooney, *Using information extraction to aid the discovery of prediction rules from text*, Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining (Boston, MA), July 2000, pp. 51–58.

[39] J. R. Quinlan, *C4.5: Programs for machine learning*, Morgan Kaufmann Publishers, San Mateo, California, 1993.

[40] J. Ross Quinlan, *Learning first-order definitions of functions*, Journal of Artificial Intelligence Research **5** (1996), 139–161.

[41] I. Schwab, A. Kobsa, and I. Koychev, *Learning about users from observation*, Adaptive User Interfaces: Papers from the 2000 AAAI Spring Symposium (Menlo Park, CA, USA), AAAI Press, 2000.

[42] I. Schwab, W. Pohl, and I. Koychev, *Learning to recommend from positive evidence*, Proceedings of the 2000 International Conference on Intelligent User Interfaces (New Orleans, LA, USA), 2000, pp. 241–248.

[43] Maria Virvou, Victoria Tsiriga, and M. Moundridou, *Adaptive navigation support in a web-based software engineering course*, Proceedings of the 2nd International Conference on Technology in Teaching and Learning in Higher Education, 2001, pp. 333–338.

[44] Gerhard Weber and Marcus Specht, *User modeling and adaptive navigation support in WWW-based tutoring systems*, Proceedings of the Sixth International Conference of User Modeling, UM'97 (A. Jameson, C. Paris, and C. Tasso, eds.), Springer-Verlag, 1997, pp. 289–300.

[45] Yongqiao Xiao and Margaret H. Dunham, *Efficient mining of traversal patterns*, Data and Knowledge Engineering **39** (2001), no. 2, 191–214.

[46] Jie Zhang and Ali Ghorbani, *The reconstruction of user sessions from a server log using improved time-oriented heuristics*, Proceedings of Communication Network and Services Research (CNSR) 2004 Conference (Fredericton, NB, Canada), May 19–21 2004, pp. 315–322.

[47] Jie Zhang and Ali A. Ghorbani, *A generic user modeling server for adaptive web systems*, MITACS 5th Annual Conference (poster) (Halifax, Canada), June 9–12 2004.

[48] I. Zukerman and D. Albrecht, *Predictive statistical models for user modeling*, User Modeling and User-Adapted Interaction **11** (2001), 5–18.

Appendix A

Implementation Code

Table A.1 presents the components that have been implemented and the classes whose source code is released. The classes whose source code is released are marked by ✓.

A.1 User Profile Manager

A.1.1 CTupm Class

```
import java.net.*;
import java.util.*;

public class CTupm {

    //Global variables for the network connection;
    public static byte[] regIP = {(byte)127,(byte)0,(byte)0,(byte)1};
    public static int regPort = 4444;

    public static void main(String[] args)throws Exception{
        //initialize all the parameters which have been defined as global variables
        if(args.length == 1){
            System.out.println("Command line: CTupm [regIP port] [dburl username password]");
            System.out.println("Example: CTupm 131.202.240.228 4444 ias.cs.unb.ca/adwert **** ****");
            System.out.println("[regIP] is the IP address of the machine that CTs will register to");
            System.out.println("[port] is the port number of the machine that CTs will register to");
            System.exit(1);
        }else if(args.length >= 2){
            StringTokenizer token = new StringTokenizer(args[0], ".");
            for(int i=0; i<regIP.length && token.hasMoreTokens(); i++){
                regIP[i] = (byte)(Integer.parseInt(token.nextToken()));
            }
            regPort = Integer.parseInt(args[1]);
        }
        CTupm ct = new CTupm();
        ct.startCT();
    }

    public void startCT() throws Exception{
        //get Modeling Interface address for registration
        InetAddress regAddr = null;
        try {
            regAddr = InetAddress.getByAddress(regIP);
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }

        //get local machine address for running all the conceptual tasks
        InetAddress ctServiceAddr = null;
        try {
```

Table A.1: Implemented Components and Released Code

System Components	Class Name	Whether Released
Authoring Tool	Authoring	
	UMPanel	
	GMPanel	
Modeling Interface	ModelingInterface	
User Profile Manager	CTupm	✓
	UPManager	✓
	Reply	✓
	DBManager	
Usage Group Handler	UGHandler	✓
	Clustering	
	DBManager	✓
Association Miner	AssociationMiner	✓
	DBManager	
	Mining	
	Rule	
Inference Engine	Inference	
Recommender	CTrecommender	✓
	DBManager	
	Recommender	✓
	Reply	✓
Profile Editor	Viewer	
	Modifier	

```
        ctServiceAddr = InetAddress.getLocalHost();
    } catch (UnknownHostException e) {
        e.printStackTrace();
    }

    int upmTaskID = 6666; // ID for the user profile manager
    int upmServicePort = 6666; //server port for the user profile manager
    register(upmTaskID, regAddr, regPort, ctServiceAddr, upmServicePort);//register user profile manager
    UPManager upManager = new UPManager(upmServicePort);//start user profile manager thread
    upManager.start();
}

//method for registration of the User Profile Manager
public void register(int seqID, InetAddress regAddr, int regPort, InetAddress ctServiceAddr, int ctServicePort){
    byte[] buffer = new byte[8];
    try{
        //connect to the Modeling Interface
        InetSocketAddress regPos = new InetSocketAddress(regAddr, regPort);
        DatagramPacket packet = new DatagramPacket(buffer, buffer.length, regPos);
        DatagramSocket socket = new DatagramSocket();
        socket.connect(regPos);

        //data of registration information
        buffer[0] = (byte)(seqID/256);
        buffer[1] = (byte)(seqID % 256);
        byte[] lad = ctServiceAddr.getAddress();
        System.arraycopy(lad, 0, buffer, 2, 4);
        buffer[6] = (byte)(ctServicePort / 256);
        buffer[7] = (byte)(ctServicePort % 256);

        //send registration request
```

```
            socket.send(packet);
            System.out.println("Registration of the UPM: (" + (InetSocketAddress)socket.getLocalSocketAddress() + " -> " + regPos + ")");
            socket.close();
        }catch (Exception e){
            System.out.println("Registration Error: " + e.getMessage());
        }
    }
}
```

A.1.2 UPManager Class

```
import interpreter.Interpreter;
import java.net.*;

public class UPManager extends Thread {

    private final int maxPacketLength = 1024;
    private final String vocabFile = "c:\\jay\\awwg\\data\\gums\\description\\umdescvocab.xml";
    private final String vocabURI = "http://ias.cs.unb.ca/~jay/UMDescVocab";
    private final String defaultFile = "c:\\jay\\awwg\\data\\gums\\description\\updefault.xml";
    private final String defaultURI = "http://ias.cs.unb.ca/~jay/UPDefault";
    private final String defaultResource = "http://ias.cs.unb.ca/~jay/UPDefault#defaultUserProfile";
    private final String key = "userID";
    private Vector umTables;
    private int serverPort;
    private DatagramSocket sck;
    private static int count = 0;

    public UPManager(int serverPort) throws Exception{
        super("Start User Profile Manager...");
        this.serverPort = serverPort;
        init();
    }

    public void init() throws Exception{
        /** initializatin for the user profile manager **/
        //create tables object in memory and used for all user profiles;
        Interpreter interpreter = new Interpreter(vocabFile, vocabURI, defaultFile,
                                  defaultURI, defaultResource, key);
        interpreter.readRDF();

        //create tables for user model in the database
        umTables = interpreter.convert();
        interpreter.insertResource(umTables);
        //interpreter.printTable(umTables);
        interpreter.createTables(umTables);

        //insert a default user profile
        interpreter.insertDefault(umTables);
    }

    public void stopThread() {
        sck.close();
    }

    public void run() {
        System.out.println("User Profile Manager is spawned.");
        try {
            sck = new DatagramSocket(serverPort);
            while (true) {
                count++;

                //get the accepted packet
                byte[] buffer = new byte[maxPacketLength];
                DatagramPacket packet = new DatagramPacket(buffer, maxPacketLength);
                sck.receive(packet);
                byte[] data = packet.getData();

                //launch a new reply thread to handle the request and generate reponse
                Reply r = new Reply(count, (InetSocketAddress) (packet.getSocketAddress()),
                                    data, packet.getLength(), umTables);
                r.start();
            }
        } catch (Exception e) {
            System.out.println("Warning (user profile manager): " + e.getMessage());
        }
        if (!sck.isClosed()) {
            sck.close();
        }
    }
}
```

A.1.3 Reply Class

```
import java.util.*;
import java.net.*;

public class Reply extends Thread {
    //the value of e
    private final double e = 2.718;
    //the value of alpha to calculate the weight of previous visits
    private final double alpha = 1.0/((double)(1000*60*60*24*7));
    private final int timeOut = 10 * 1000; //set time out
    private InetSocketAddress interPos;
    private byte[] data;
    private DatagramSocket sck;
    private int dataLen;
    private Vector tables;
    private UPManager manager;
    private int count;

    public Reply(int rcount, InetSocketAddress interPos, byte[] rdata, int rdataLen, Vector rtables) {
        super("Reply");
        this.count = rcount;
        this.interPos = interPos;
        this.data = rdata;
        this.dataLen = rdataLen;
        this.tables = rtables;
        manager = new UPManager(rtables);
    }

    public void stopReply() {
        sck.close();
    }

    public void run() {
        try {
            System.out.print("user profile manager request #" + count + " ");
            System.out.print ("length: " + dataLen +"   ");
            System.out.print("Package received: ");
            for(int i = 0; i < dataLen; i++)
                System.out.print(data[i] + " ");

            sck = new DatagramSocket();
            sck.connect(interPos);
            byte[] rdata = replyHandler(data, dataLen);
            DatagramPacket packet = new DatagramPacket(rdata, rdata.length, interPos);
            sck.setSoTimeout(timeOut);
            sck.send(packet);
            sck.disconnect();
            if (!sck.isClosed())
                sck.close();
        } catch (Exception e) {
            System.out.println("Warning (user profile manager Reply): " + e.getMessage());
        }
        System.out.println("Reply operator " + count + " terminated.\n");
    }

    //a method to handle the errors
    public byte[] errorHandler(byte[] rdata, int rdataLen, String error)
        throws Exception {
        byte[] response = response = new byte[2 + error.length()];
        response[0] = rdata[0];
        response[1] = rdata[1];
        for (int i = 0; i < error.length(); i++) {
            response[i + 2] = (byte) (error.charAt(i));
        }
        System.out.println(error);
        return response;
    }

    //a method to handle the request and generate reponse
    public byte[] replyHandler(byte[] rdata, int rdataLen) throws Exception {
        byte[] response = null;
        if (rdataLen < 4) {
            String error = "message length should be longer 3";
            response = errorHandler(rdata, rdataLen, error);
        } else {
            int reqID = (int) rdata[2];
            String parameters = "";
            for (int i = 3; i < rdataLen; i++) {
                parameters = parameters + (char) rdata[i];
            }


            System.out.println("seq ID for user profile manager request #"
                + count + " " + ((int)(rdata[0]) + (rdata[1] >= 0 ? rdata[1] : rdata[1] + 256)));
```

```
System.out.println(reqID + " " + parameters);

if (reqID == 1) {
    //check if the user is a existing user
    response = new byte[3];
    response[0] = rdata[0];
    response[1] = rdata[1];
    response[2] = (byte) ((manager.exist(parameters)+"").charAt(0));
} else if (reqID == 2) {
    //check if the login username and password are correct.
    StringTokenizer token1 = new StringTokenizer(parameters, "#");
    String userName = token1.nextToken();
    String paraName = token1.nextToken();
    String paraValue = token1.nextToken();
    response = new byte[3];
    response[0] = rdata[0];
    response[1] = rdata[1];
    response[2] = (byte) ((manager.check(userName, paraValue)+"").charAt(0));
} else if (reqID == 3) {
    //create a new user profile for the user
    response = new byte[3];
    response[0] = rdata[0];
    response[1] = rdata[1];
    response[2] = (byte) ((manager.create(parameters)+"").charAt(0));
} else if (reqID == 4) {
    //delete the user's profile
    response = new byte[3];
    response[0] = rdata[0];
    response[1] = rdata[1];
    response[2] = (byte) ((manager.delete(parameters)+"").charAt(0));
} else if (reqID == 5) {
    //update value for a property
    Vector knownPair = new Vector();
    StringTokenizer token1 = new StringTokenizer(parameters, "#");
    String userName = token1.nextToken();
    knownPair.add("userID#" + userName);
    while (token1.hasMoreTokens()) {
        knownPair.add(token1.nextToken() + "#" + token1.nextToken());
    }
    String lastPair =
        (String) (knownPair.elementAt(knownPair.size() - 1));
    StringTokenizer token2 = new StringTokenizer(lastPair, "#");
    String pairName = token2.nextToken();
    String pairValue = token2.nextToken();
    knownPair.removeElementAt(knownPair.size() - 1);
    response = new byte[3];
    response[0] = rdata[0];
    response[1] = rdata[1];
    response[2] =
        (byte) ((manager.update(knownPair, pairName, pairValue)+"").charAt(0));
} else if (reqID == 6) {
    //retrieve value for a property
    Vector knownPair = new Vector();
    StringTokenizer token1 = new StringTokenizer(parameters, "#");
    String userName = token1.nextToken();
    knownPair.add("userID#" + userName);
    while (token1.hasMoreTokens()) {
        String prop = token1.nextToken();
        if(!token1.hasMoreTokens()){
            knownPair.add(prop);
        }else{
            knownPair.add(prop + "#" + token1.nextToken());
        }
    }
    String propName =
        (String) (knownPair.elementAt(knownPair.size() - 1));
    knownPair.removeElementAt(knownPair.size() - 1);
    String value = manager.retrieve(knownPair, propName);
    response = new byte[2 + value.length()];
    response[0] = rdata[0];
    response[1] = rdata[1];
    for (int i = 0; i < value.length(); i++) {
        response[i + 2] = (byte) (value.charAt(i));
    }
} else if (reqID == 7) {
    //reset property value
    Vector knownPair = new Vector();
    StringTokenizer token1 = new StringTokenizer(parameters, "#");
    String userName = token1.nextToken();
    knownPair.add("userID#" + userName);
    while (token1.hasMoreTokens()) {
        String prop = token1.nextToken();
        if(!token1.hasMoreTokens()){
            knownPair.add(prop);
        }else{
```

```
            knownPair.add(prop + "#" + token1.nextToken());
        }
    }
    String propName =
        (String) (knownPair.elementAt(knownPair.size() - 1));
    knownPair.removeElementAt(knownPair.size() - 1);
    response = new byte[3];
    response[0] = rdata[0];
    response[1] = rdata[1];
    response[2] = (byte) ((manager.reset(knownPair, propName)+"").charAt(0));
} else if(reqID == 8) {
    //add value for the user;
    Vector knownPair = new Vector();
    StringTokenizer token = new StringTokenizer(parameters, "#");
    String userName = token.nextToken();
    knownPair.add("userID#" + userName);
    while (token.hasMoreTokens()) {
        knownPair.add(token.nextToken() + "#" + token.nextToken());
    }
    response = new byte[3];
    response[0] = rdata[0];
    response[1] = rdata[1];
    response[2] = (byte) ((manager.addValue(knownPair)+"").charAt(0));
}else if(reqID == 9){
    //check if the value exists
    Vector knownPair = new Vector();
    StringTokenizer token = new StringTokenizer(parameters, "#");
    String userName = token.nextToken();
    knownPair.add("userID#" + userName);
    while (token.hasMoreTokens()) {
        knownPair.add(token.nextToken() + "#" + token.nextToken());
    }
    response = new byte[3];
    response[0] = rdata[0];
    response[1] = rdata[1];
    response[2] = (byte) ((manager.checkValue(knownPair)+"").charAt(0));
}else if (reqID == 10) {
    //get content of the user model
    String value = manager.retrieveAll(parameters);
    response = new byte[2 + value.length()];
    response[0] = rdata[0];
    response[1] = rdata[1];
    for (int i = 0; i < value.length(); i++) {
        response[i + 2] = (byte) (value.charAt(i));
    }
}else if(reqID == 11) {
    //add value for the user;
    Vector knownPair = new Vector();
    long newDate = (new Date()).getTime();
    System.out.println("new Date: " + newDate);
    parameters = parameters + "#date#" + newDate;
    StringTokenizer token = new StringTokenizer(parameters, "#");
    String userName = token.nextToken();
    knownPair.add("userID#" + userName);
    while (token.hasMoreTokens()) {
        knownPair.add(token.nextToken() + "#" + token.nextToken());
    }

    response = new byte[3];
    response[0] = rdata[0];
    response[1] = rdata[1];

    response[2] = (byte) ((manager.addValue(knownPair)+"").charAt(0));

    /**update the interested topics table**/
    //get the value of newsID from the received message
    String newsID = manager.getValue(knownPair, "newsID");

    //get the category information of the news
    String category = manager.getCategory(newsID);

    //retrieve the value of interests value for the category
    Vector known = new Vector();
    known.add("userID#" + userName);
    String interestValue = manager.retrieve(known, category);

    //retrieve the value of date for the category
    long oldDate = Long.parseLong(manager.retrieve(known, "interestDate"));
    System.out.println("old Date: " + oldDate);

    //calculate the time difference between the old date and current time
    double timeDiff = ((double)(newDate-oldDate));
    System.out.println("time:" + timeDiff);

    //calculate the new value from the current added value and the weighted old value
    String newValue = (1.0 + (Double.parseDouble(interestValue))*(Math.pow(e, -(alpha*timeDiff))))+"";
```

```
                //update value of the interests value and the date of updating
                int successful = manager.update(known, category, newValue);
                successful = manager.update(known, "interestDate", newDate+"");
            }else {
                String error = "invalid reqID for the user profile manager";
                response = errorHandler(rdata, rdataLen, error);
            }

            String result = "reply for the user profile manager request #" + count + " is: ";
            for (int i = 2; i < response.length; i++) {
                result = result + (char) response[i];
            }
            System.out.println(result);
        }
        return response;
    }
}
```

A.2 Usage Group Handler

A.2.1 UGHandler Class

```
import java.util.*;
import interpreter.*;

import ughandler.clustering.Clustering;
import ughandler.kmeans.*;

public class UGHandler extends Thread{

    private Vector groupModel;
    final private int initialClusterNum = 5;
    final private double simThreshold = 0.1;
    //the percent threshold of a group population
    final private double percent = 0.5;

    private String interestsFile = "";
    private String centerFile = "";

    final private boolean kMeansPlus = true;
    //final private boolean kMeansPlus = false;

    final private double support = 0.4;
    private DBManagement dbmanager;
    //group model structure
    private Vector gmTables;
    //the array to store the categories of concepts
    private String[] categories;
    private FileManagement filemanager;
    private Clustering clustering;
    private kMeans kmeans;
    private int[] popuList;

    public UGHandler(Vector tables) {
        if(kMeansPlus){
            interestsFile = "c:\\jay\\awwg\\data\\gums\\ughandler\\clustering\\data\\train_input.data";
            centerFile = "c:\\jay\\awwg\\data\\gums\\ughandler\\clustering\\data\\centerFile.data";
        }else{
            interestsFile = "c:\\jay\\awwg\\data\\gums\\ughandler\\kmeans\\data\\train_input.data";
            centerFile = "c:\\jay\\awwg\\data\\gums\\ughandler\\kmeans\\data\\centerFile.data";
        }

        gmTables = tables;
        categories = getCategories();
        dbmanager = new DBManagement(categories);
        filemanager = new FileManagement(interestsFile, centerFile);
        clustering = new Clustering(interestsFile, centerFile);
        kmeans = new kMeans(initialClusterNum, interestsFile, centerFile);
    }

    //group users once after one week
    public void run(){
        long previousTime = new Date().getTime();
        try {
            while (true) {
                long currentTime = new Date().getTime();
                if(currentTime - 3600000*24*7 >= previousTime){
                    //grouping after a period of time
                    grouping();
                    findPop();
```

```
            }else if(dbmanager.exceed(percent)){
                //if the new population exceeds a threshold, do grouping
                grouping();
                findPop();
            }
            previousTime = new Date().getTime();
            //check after a period of time
            sleep(60000);
        }
    } catch (Exception e) {
        System.out.println("Retrieve news thread error: " + e.getMessage());
    }
}

//do the clustering
public void grouping() throws Exception {
    Vector interests = dbmanager.retrieveData();
    Vector normInte = new Vector();

    //normalize the interests values
    for(int i=0; i<interests.size(); i++){
        Interests userInterest = (Interests)(interests.elementAt(i));
        normInte.add(new Interests(userInterest.getName(),normalize(userInterest.getInterests())));
    }

    //write into the interests file
    filemanager.writeFile(normInte);

    if(kMeansPlus){
        //clustering the users based on their interests value
        clustering.run(initialClusterNum);
        //get population of each group
        popuList = clustering.getPopu();
    }else{
        //clustering the users based on their interests value
        kmeans.readData();
        kmeans.runKMeans();
        kmeans.writeCenters();
        popuList = kmeans.getPopu();
        for(int i=0; i<popuList.length; i++){
            System.out.println(popuList[i]);
        }
    }

    //read group's center information from the center file
    Vector groups = filemanager.readFile();

    //insert groups into the database
    Vector newGroups = new Vector();
    Vector popu = new Vector();
    for(int i=0; i<groups.size(); i++){
        if(popuList[i] > 0){
            newGroups.add((String[])(groups.elementAt(i)));
            popu.add(popuList[i]+"");
        }
    }
    dbmanager.insertGroup(newGroups, popu);

    //update the user's group information
    updateUPs(interests, newGroups);
    System.out.println("grouping is done successfully!");
}

//get categories from the table structure for the rdf description of group model
private String[] getCategories(){
    Vector properties = null;
    Vector categories = new Vector();
    boolean found = false;
    for(int i=0; i<gmTables.size() && !found; i++){
        Table t = (Table)(gmTables.elementAt(i));
        if(t.getName().equals("GroupModel")){
            found = true;
            properties = t.getProperties();
        }
    }

    for(int i=0; i<properties.size(); i++){
        interpreter.Properties property = (interpreter.Properties)(properties.elementAt(i));
        if(!property.getName().equals("groupID") && !property.getName().equals("groupPopu")
            && !property.getName().equals("newPopu")){
            categories.add(property.getName());
        }
    }
    String[] cateArray = new String[categories.size()];
    for(int i=0; i<categories.size(); i++){
        cateArray[i] = (String)(categories.elementAt(i));
```

```
        //System.out.println("category: " + cateArray[i]);
    }
    return cateArray;
}

//get population of a group
public int getPopu(){
    int popu = 0;
    for(int i=0; i<popuList.length; i++){
        popu = popu + popuList[i];
    }
    return popu;
}

//normalize the interests value
private double[] normalize(double[] interests){
    int length = interests.length;
    double[] normalizedInterests = new double[length];
    double sum = 0;
    for(int i=0; i<length; i++){
        sum = sum + interests[i]*interests[i];
    }
    double vecLen = Math.sqrt(sum);
    for(int i=0; i<length; i++){
        if(vecLen == 0){
            normalizedInterests[i] = 0;
        }else{
            normalizedInterests[i] = interests[i]/vecLen;
        }
    }
    return normalizedInterests;
}

//update user groups information
private void updateUPs(Vector interests, Vector groups)throws Exception{

    dbmanager.updateUPs(interests, groups);
}

//assign a user to a group
private int assignGroup(double[] userInterests, Vector centers){
    double similarity = -1.0;
    int group = -1;
    for (int i = 0; i < centers.size(); i++) {
        String[] center = (String[])(centers.elementAt(i));
        double sim = similarity(userInterests, center);
        if (similarity <= sim) {
            group = i;
            similarity = sim;
        }
    }
    return group;
}

//assign a user into a existing group
private int assignExistGroup(double[] userInterests, Vector centers)throws Exception{
    double similarity = -1.0;
    int group = -1;
    boolean below = true;
    for (int i = 0; i < centers.size(); i++) {
        String[] center = (String[])(centers.elementAt(i));
        double sim = similarity(userInterests, center);
        //System.out.println(sim);
        if(sim >= simThreshold){
            below = false;
        }
        if (similarity <= sim) {
            group = i;
            similarity = sim;
        }
    }
    if(below == true){
        group = centers.size();
        centers.add(userInterests);
        dbmanager.insertCenter(group, userInterests, 1);
    }else{
        dbmanager.updateGroup(userInterests, (String[])(centers.elementAt(group)), group);
    }
    return group;
}

//assign user into a group
public int groupMatch(String userID) throws Exception {
    double[] userInterests = dbmanager.retrieveUserData(userID);
    userInterests = normalize(userInterests);
    Vector centers = dbmanager.retrieveCenters();
```

```
    int group = assignExistGroup(userInterests, centers);
    dbmanager.updateNewPopu(group);
    dbmanager.updateUP(group, userID);
    return 1;
}

//a way to calculate similarity
private double similarity(double[] user, String[] center) {
    double similarity = 0.0;
    double norm = 0.0;
    for (int i = 0; i < user.length; i++) {
        similarity = similarity + user[i] * (Double.parseDouble(center[i]));
    }
    return similarity;
}

//another way to calculate similarity
/**
private double similarity(double[] user, String[] center){
    double similarity = 0.0;
    for(int i=0; i<user.length; i++){
        similarity = similarity + Math.abs(user[i]-Double.parseDouble(center[i]));
    }
    similarity = 1 - similarity/center.length;
    return similarity;
}
**/

public void findPop() throws Exception {
    int numOfGroups = dbmanager.getNumOfGroups();
    Vector[] groups = new Vector[numOfGroups];
    for (int i = 0; i < numOfGroups; i++) {
        groups[i] = dbmanager.getUsers(i);
        groups[i] = dbmanager.getUserProps(groups[i]);
    }

    for(int i=0; i<groups.length; i++){
        System.out.println("group#" + i);
        try{
            Writer fw = new Writer("c:\\jay\\evaluation\\dataset2\\test" + i + ".txt");

            for(int j=0; j<groups[i].size(); j++){
                Vector values = (Vector)(groups[i].elementAt(j));
                String vs = "";
                for(int k=0; k<values.size(); k++){
                    vs = vs + (String)(values.elementAt(k)) + "#";
                }
                fw.write(vs);
            }
        }catch (Exception e){

        }

    }

    for(int i=0; i<numOfGroups; i++){
        Vector values = (Vector)(groups[i].elementAt(0));
        int propNum = values.size();
        System.out.println(propNum);
        Vector popvalues = new Vector();
        for(int j=0; j<propNum; j++){
            Hashtable vhash = new Hashtable();
            for(int k=0; k<groups[i].size(); k++){
                Vector vs = (Vector)(groups[i].elementAt(k));
                String v = (String)(vs.elementAt(j));
                if(vhash.containsKey(v)&&!v.equals(" ")){
                    int n = Integer.parseInt((String)(vhash.get(v)));
                    n++;
                    vhash.remove(v);
                    vhash.put(v,n+"");
                }else if(!v.equals(" ")){
                    vhash.put(v,"1");
                }
            }

            int total = 0;
            int max = 0;
            String maxProp = "";
            Object[] keys = vhash.keySet().toArray();
            for(int k=0; k<keys.length; k++){
                int n = Integer.parseInt((String)(vhash.get(keys[k])));
                total = total + n;
                System.out.println("group" + i);
                System.out.println(keys[k] + "   " + n);
                if(n>=max){
                    maxProp = (String)(keys[k]);
```

```
                    max = n;
                }
            }
            if(max >= total*support)
                popvalues.add(maxProp);
        }

        if(popvalues.size()>0){
            Hashtable allPop = new Hashtable();
            for(int j=0; j<popvalues.size(); j++){
                StringTokenizer token = new StringTokenizer((String)(popvalues.elementAt(j)), "=");
                allPop.put(token.nextToken(), token.nextToken());
            }

            dbmanager.insertPop(i, allPop);
        }
    }
}
```

A.2.2 DBManager Class

```
import java.sql.*;
import java.util.*;

public class DBManager {
    //database management for the group handler an array to store the sections' name
    private String[] sections;
    private DBConnection connection;

    public DBManager(String[] sections){
        connection = new DBConnection();
        this.sections = sections;
    }

    //insert center infor into the databse
    public void insertCenter(int groupID, double[] userInterests, int groupPopu) throws Exception{
        //construct a query
        String query = "insert into GroupModel (GroupID, groupPopu, newPopu,";
        for (int i = 0; i < sections.length; i++) {
            if (i != sections.length - 1)
                query = query + sections[i] + ",";
            else
                query = query + sections[i] + ") values (";
        }
        int rs = -1;
        String values = groupID + "," + groupPopu + ",0,";
        for (int j = 0; j < userInterests.length; j++) {
            if (j != userInterests.length - 1)
                values = values + "'" + userInterests[j] + "',";
            else
            values = values + "'" + userInterests[j] + "')";
        }

        //open database connection and execute query
        Connection conn = connection.openConnection();
        PreparedStatement pstmt = null;
        pstmt = conn.prepareStatement(query + values);
        rs = pstmt.executeUpdate();

        //close connection
        pstmt.close();
        connection.closeConnection(conn);
    }

    //update the groups with new population and new centers
    public void updateGroup(double[] userInterests, String[] center, int group)throws Exception{
        //get the old population of the group
        String query = "select groupPopu from GroupModel where groupID = " + group;
        Connection conn = connection.openConnection();
        PreparedStatement pstmt = conn.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery();
        int popu = -1;
        if (rs.next()) {
            popu = rs.getInt(1);
        }
        rs.close();
        String[] newCenter = new String[center.length];
        for(int i=0; i<newCenter.length; i++){
            newCenter[i] = ((Double.parseDouble(center[i])*popu+userInterests[i])/(popu+1))+"";
        }

        //update groups with new centers and new population
```

```
    query = "update GroupModel set groupPopu = " + (popu+1) + ", ";
    for (int i = 0; i < sections.length; i++) {
        if (i != sections.length - 1)
            query = query + sections[i] + "= '" + newCenter[i] + "', ";
        else
            query = query + sections[i] + "= '" + newCenter[i] + "';;
    }
    query = query + " where groupID = " + group + ";";
    pstmt = conn.prepareStatement(query);
    int rs2 = pstmt.executeUpdate();

    //close connection
    pstmt.close();
    connection.closeConnection(conn);
}

//insert groups into the group model database
public void insertGroup(Vector groups, Vector popuList) throws Exception {
    //delete the old groups stored in the database
    String query = "delete from GroupModel where groupID <> -1;";
    PreparedStatement pstmt = null;
    Connection conn = connection.openConnection();
    pstmt = conn.prepareStatement(query);
    int rs = -1;
    rs = pstmt.executeUpdate();

    //insert new groups into the database
    query = "insert into GroupModel (groupID, groupPopu, newPopu,";
    for (int i = 0; i < sections.length; i++) {
        if (i != sections.length - 1)
            query = query + sections[i] + ",";
        else
            query = query + sections[i] + ") values (";
    }

    for (int i = 0; i < groups.size(); i++) {
        String[] center = (String[]) (groups.elementAt(i));
        String values = i + "," + Integer.parseInt((String)(popuList.elementAt(i))) + ",0,";

        for(int j=0; j<center.length; j++){
            System.out.println(j + ": " + center[j]);
        }
        for (int j = 0; j < center.length; j++) {
            if (j != center.length - 1)
                values = values + "'" + center[j] + "',";
            else
                values = values + "'" + center[j] + "')";
        }
        System.out.println(query+values);
        pstmt = conn.prepareStatement(query + values);
        rs = pstmt.executeUpdate();
    }

    //close connection
    pstmt.close();
    connection.closeConnection(conn);
}

public void updateUPs(Vector interests, Vector groups)throws Exception{
    Connection conn = connection.openConnection();
    for(int i=0; i<interests.size(); i++){
        Interests interest = (Interests)(interests.elementAt(i));
        double[] userInterests = interest.getInterests();
        double[] normUserInterests = normalize(userInterests);
        int group = assignGroup(normUserInterests, groups);
        updateUP(group, interest.getName(), conn);
    }
    conn.close();
}

//normalize the interests value
private double[] normalize(double[] interests){
    int length = interests.length;
    double[] normalizedInterests = new double[length];
    double sum = 0;
    for(int i=0; i<length; i++){
        sum = sum + interests[i]*interests[i];
    }
    double vecLen = Math.sqrt(sum);
    for(int i=0; i<length; i++){
        if(vecLen == 0){
            normalizedInterests[i] = 0;
        }else{
            normalizedInterests[i] = interests[i]/vecLen;
        }
    }
```

```
    return normalizedInterests;
}

//a way to calculate similarity
private double similarity(double[] user, String[] center) {
    double similarity = 0.0;
    double norm = 0.0;
    for (int i = 0; i < user.length; i++) {
        similarity = similarity + user[i] * (Double.parseDouble(center[i]));
    }
    return similarity;
}

//assign a user to a group
private int assignGroup(double[] userInterests, Vector centers){
    double similarity = -1.0;
    int group = -1;
    for (int i = 0; i < centers.size(); i++) {
        String[] center = (String[])(centers.elementAt(i));
        double sim = similarity(userInterests, center);
        if (similarity <= sim) {
            group = i;
            similarity = sim;
        }
    }
    return group;
}

//get information from the groups
public Vector retrieveData() throws Exception {
    Vector interests = new Vector();

    //construct query
    String query = "select userID, ";
    for (int i = 0; i < sections.length; i++) {
        if (i != sections.length - 1)
            query = query + sections[i] + ", ";
        else
            query = query + sections[i] + " ";
    }
    query = query + "from InterestedProducts;";

    //open connection and execute query
    Connection conn = connection.openConnection();
    PreparedStatement pstmt = conn.prepareStatement(query);
    ResultSet rs = pstmt.executeQuery();
    while (rs.next()) {
        double[] intrerest = new double[sections.length];
        for (int i = 0; i < sections.length; i++) {
            intrerest[i] = rs.getDouble(i + 2);
        }
        if (!rs.getString(1).equals("default"))
            interests.add(new Interests(rs.getString(1), intrerest));
    }

    //close connection
    pstmt.close();
    rs.close();
    connection.closeConnection(conn);

    return interests;
}

//retrieve user date from database
public double[] retrieveUserData(String userID)throws Exception{
    //construct query
    double[] userInterests = new double[sections.length];
    String query = "select ";
    for (int i = 0; i < sections.length; i++) {
        if (i != sections.length - 1)
            query = query + sections[i] + ", ";
        else
            query = query + sections[i] + " ";
    }
    query = query + "from InterestedProducts where userID = '" + userID + "';";

    //open connection to execute the query
    Connection conn = connection.openConnection();
    PreparedStatement pstmt = conn.prepareStatement(query);
    ResultSet rs = pstmt.executeQuery();
    if (rs.next()) {
        for (int i = 0; i < sections.length; i++) {
        userInterests[i] = (double)(rs.getInt(i + 1));
        }
    }
```

```
    //close connection
    rs.close();
    pstmt.close();
    connection.closeConnection(conn);

    return userInterests;
}

//get center information
public Vector retrieveCenters()throws Exception{
    Vector centers = new Vector();
    //construct query
    String query = "select groupID, ";
    for (int i = 0; i < sections.length; i++) {
        if (i != sections.length - 1)
            query = query + sections[i] + ", ";
        else
            query = query + sections[i] + " ";
    }
    query = query + "from GroupModel where groupID <> -1;";

    //open connection to execute query
    Connection conn = connection.openConnection();
    PreparedStatement pstmt = conn.prepareStatement(query);
    ResultSet rs = pstmt.executeQuery();
    while (rs.next()) {
        String[] center = new String[sections.length];
        for (int i = 0; i < sections.length; i++) {
            center[i] = rs.getString(i + 2);
        }
        centers.add(rs.getInt(1), center);
    }

    //close connection
    pstmt.close();
    rs.close();
    connection.closeConnection(conn);

    return centers;
}

//update user's group information
public void updateUP(int group, String userName, Connection conn)throws Exception{
    //construct query
    String query =
        "update GroupInfor set groupID = "
            + group
            + " where userID = '"
            + userName
            + "';";

    //open connection to execute query

    PreparedStatement pstmt = conn.prepareStatement(query);
    int rs = pstmt.executeUpdate();

    //close connection
    pstmt.close();
}

//update user's group information
public void updateUP(int group, String userName)throws Exception{
    //construct query
    String query =
        "update GroupInfor set groupID = "
            + group
            + " where userID = '"
            + userName
            + "';";

    //open connection to execute query
    Connection conn = connection.openConnection();
    PreparedStatement pstmt = conn.prepareStatement(query);
    int rs = pstmt.executeUpdate();

    //close connection
    pstmt.close();
    conn.close();
}

//update new population
public void updateNewPopu(int group) throws Exception{
    String query = "update GroupModel set newPopu = newPopu + 1 where groupID =" + group + ";";

    //open connection to execute query
    Connection conn = connection.openConnection();
```

```
    PreparedStatement pstmt = null;
    pstmt = conn.prepareStatement(query);
    int rs = pstmt.executeUpdate();

    //close connection
    pstmt.close();
    connection.closeConnection(conn);
}

//check if the new population exceeds certain percentation of old one or not
public boolean exceed(double percent) throws Exception{
    boolean exceed = false;

    String query = "select groupPopu, newPopu from GroupModel where groupID <> -1";

    //open connection to execute query
    Connection conn = connection.openConnection();
    PreparedStatement pstmt = conn.prepareStatement(query);
    ResultSet rs = pstmt.executeQuery();
    while (rs.next() && !exceed) {
        int groupPopu = rs.getInt(1);
        int newPopu = rs.getInt(2);
        if(newPopu >= groupPopu*percent){
            exceed = true;
        }
    }

    //close connection
    pstmt.close();
    rs.close();
    connection.closeConnection(conn);

    return exceed;
}

//get total number of groups that the system has
public int getNumOfGroups() throws Exception{
    String query = "select count(*) from GroupModel;";

    //open connection with database
    Connection conn = connection.openConnection();
    PreparedStatement pstmt = conn.prepareStatement(query);
    ResultSet numRS = pstmt.executeQuery();
    int num = 0;
    if(numRS.next()){
        num = numRS.getInt(1);
    }

    //close connection
    pstmt.close();
    numRS.close();
    connection.closeConnection(conn);

    return num-1;
}

//get all the users in the group with the groupID
public Vector getUsers(int groupID) throws Exception{
    Vector users = new Vector();
    String query = "select userID from GroupInfor where groupID = " + groupID + ";";

    Connection conn = connection.openConnection();

    PreparedStatement pstmt = conn.prepareStatement(query);
    ResultSet numRS = pstmt.executeQuery();
    ResultSet rs = pstmt.executeQuery();
    while (rs.next()) {
        String userID = rs.getString(1);
        if(!userID.equals("default")){
            users.add(userID);
        }
    }

    pstmt.close();
    numRS.close();
    connection.closeConnection(conn);
    return users;
}

public Vector getUserProps(Vector groups)throws Exception{
    String query = "show columns from DGModel;";
    Vector propName = new Vector();
    //open connection with database
    Connection conn = connection.openConnection();
    PreparedStatement pstmt = conn.prepareStatement(query);
    ResultSet rs = pstmt.executeQuery();
```

```
    while(rs.next()){
        propName.add(rs.getString(1));
    }

    Vector groupProps = new Vector();
    for (int j = 0; j < groups.size(); j++) {
        String userID = (String) (groups.elementAt(j));
        Vector propValue = new Vector();

        query = "select ";
        for(int i=0; i<propName.size(); i++){
            if(i != propName.size()-1)
                query = query + (String)(propName.elementAt(i)) + ", ";
            else
                query = query + (String)(propName.elementAt(i)) + " ";
        }
        query = query + " from DGModel where userID = '" + userID + "';";
        pstmt = conn.prepareStatement(query);
        rs = pstmt.executeQuery();
        if(rs.next()){
            for(int i=0; i<propName.size(); i++){
                String pname = (String)(propName.elementAt(i));
                if(!pname.equals("userID")){
                    String value = rs.getString(pname);


                    query = "select " + pname + " from Resource where userID = '" + userID + "';";
                    String resource = "";
                    PreparedStatement pstmt2 = conn.prepareStatement(query);
                    ResultSet rs2 = pstmt2.executeQuery();
                    if(rs2.next()){
                        resource = rs2.getString(1);
                    }
                    pstmt2.close();
                    rs2.close();

                    if(getIndex(resource) < getIndex(group)){
                        propValue.add(pname+ "=" + value);
                    }else{
                        propValue.add(" ");
                    }
                }
            }
        }

        if(propValue.size() > 0)
            groupProps.add(propValue);
    }

    //close connection
    pstmt.close();
    rs.close();

    connection.closeConnection(conn);
    return groupProps;
}

private String getResource(String userID, String pname) throws Exception{
    String query = "select " + pname + " from Resource where userID = '" + userID + "';";
    String resource = "";
    Connection conn = connection.openConnection();
    PreparedStatement pstmt = conn.prepareStatement(query);
    ResultSet rs = pstmt.executeQuery();
    if(rs.next()){
        resource = rs.getString(1);
    }
    rs.close();
    pstmt.close();
    conn.close();
    return resource;
}

private int getIndex(String resource){
    int index = -1;
    for(int i=0; i<resources.length && index==-1; i++){
        if(resource.equals(resources[i])){
            index = i;
        }
    }
    return index;
}

public Vector getPopProps()throws Exception{
    Vector popProps = new Vector();
    String query = "show columns from GroupPopValues";
    DBConnection connection = new DBConnection();
```

```
        Connection conn = connection.openConnection();
        PreparedStatement pstmt = conn.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery();
        while(rs.next()){
            String prop = rs.getString(1);
            if(!prop.equals("groupID"))
                popProps.add(prop);
        }

        rs.close();
        pstmt.close();
        conn.close();

        return popProps;
    }

    public void insertPop(int groupID, Hashtable popPropValues) throws Exception{
        Vector columns = getPopProps();
        String query = "insert into GroupPopValues (groupID, ";
        String values = ") values (" + groupID + ", ";

        for(int i=0; i<columns.size(); i++){
            String prop = (String)(columns.elementAt(i));
            String value = "";
            if(popPropValues.containsKey(prop)){
                value = (String)(popPropValues.get(prop));
            }
            if(i != columns.size()-1){
                query = query + prop + ", ";
                values = values + "'" + value + "', ";
            }else{
                query = query + prop;
                values = values + "'" + value + "'";
            }
        }

        query = query + values + ");";
        System.out.println(query);

        //open database connection and execute query
        Connection conn = connection.openConnection();
        PreparedStatement pstmt = null;
        pstmt = conn.prepareStatement(query);
        int rs = pstmt.executeUpdate();

        //close connection
        pstmt.close();
        connection.closeConnection(conn);

    }
}
```

A.3 Association Miner

A.3.1 AssociationMiner Class

```
import java.util.*;
import interpreter.*;

public class AssociationMiner extends Thread {
    //minimal support and minimal confidence
    final private double mini_support = 0.1;
    final private double mini_confidence = 0.1;

    //the weight of confidence of support to calculate priority
    final private double weight_conf = 0.5;
    final private double weight_supp = 0.5;

    //hashtable to store the mapping
    final private String mapFile = "c:\\jay\\awwg\\data\\gums\\description\\map.dat";
    private Hashtable mapping;
    private Hashtable mapIDProp;

    private DBManagement dbmanager;
    //association rule model structure
    private Vector arTables;
    private Mining mining;

    public AssociationMiner(Vector arTables) throws Exception{
        this.arTables = arTables;
        dbmanager = new DBManagement();
```

```
        mining = new Mining(mini_support, mini_confidence);
        mapping = new Hashtable();
        mapIDProp = new Hashtable();
    }

    public AssociationMiner(){
        mapping = new Hashtable();
        mapIDProp = new Hashtable();
    }

    public Hashtable getMap(){
        return mapping;
    }

    public Hashtable getMapIDProp(){
        return mapIDProp;
    }

    public void read(){
        Reader reader = new Reader(mapFile);
        String line = "";
        while((line = reader.readLine())!=null && line != ""){
            StringTokenizer token = new StringTokenizer(line, "#");
            String id = token.nextToken();
            String prop = token.nextToken();
            String value = token.nextToken();
            mapping.put(prop+"#"+value, id);
            mapIDProp.put(id, prop+"#"+value);
        }
    }

    //do mining once in a period of time
    public void run() {
        try {
            while (true) {
                sleep(3600000 * 24 * 2);
                mining();
            }
        } catch (Exception e) {
            System.out.println("Mining thread error: " + e.getMessage());
        }
    }

    //mining process
    public void mining() {
        try {
            //retrieve user's properties
            Vector props = dbmanager.getProps(mapping);

            //delete all the old association rules;
            dbmanager.deleteRules();

            //mine association rules and insert the mined rules into the database
            Vector freSets = mining.apriori(props);
            Vector rules = mining.generateRules(freSets);
            System.out.println(rules.size());
            for(int j=0; j<rules.size(); j++){
                Rule rule = (Rule)(rules.elementAt(j));
                String output = " Rule#" + j + ": " + rule.getFirst() + " -> " + rule.getSec() + " (" + rule.getConf()
                    + ", " + rule.getSup() + ")";
                System.out.println(output);
            }
            if(rules.size() >0 ){
                dbmanager.insertRules(rules, weight_conf, weight_supp);
            }
        } catch (Exception e) {
            System.out.println("Mining error: " + e.getMessage());
        }
    }
}
```

A.4 Recommender

A.4.1 CTrecommender Class

```
import java.net.*; import java.util.*;

public class CTrecommender{

    //Global variables for the network connection;
    public static byte[] regIP = {(byte)127,(byte)0,(byte)0,(byte)1};
```

```
    public static int regPort = 4444;

    public static void main(String[] args)throws Exception{
        //initialize all the parameters which have been defined as global variables
        if(args.length == 1){
            System.out.println("Command line: CTrecommender [regIP port] [dburl username password]");
            System.out.println("Example: CTrecommender 131.202.240.228 4444");
            System.out.println("[regIP] is the IP address of the machine that CTs will register to");
            System.out.println("[port] is the port number of the machine that CTs will register to");
            System.exit(1);
        }else if(args.length >= 2){
            StringTokenizer token = new StringTokenizer(args[0], ".");
            for(int i=0; i<regIP.length && token.hasMoreTokens(); i++){
                regIP[i] = (byte)(Integer.parseInt(token.nextToken()));
            }
            regPort = Integer.parseInt(args[1]);
        }

        //launch a recommender thread
        CTrecommender ct = new CTrecommender();
        ct.startCT();
    }

    public void startCT() throws Exception{
        //get modeling interface address for registration
        InetAddress regAddr = null;
        try {
            regAddr = InetAddress.getByAddress(regIP);
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }

        //get local machine address for running all the components
        InetAddress ctServiceAddr = null;
        try {
            ctServiceAddr = InetAddress.getLocalHost();
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }

        //task ID for the Recommender
        int amTaskID = 8888;
        //service port number
        int amServicePort = 8888;

        //register the association miner
        register(amTaskID, regAddr, regPort, ctServiceAddr, amServicePort);

        //start association miner thread
        CTassocMiner miner = new CTassocMiner(amServicePort);
        miner.start();

    }

    //method for registration of the Recommender
    public void register(int seqID, InetAddress regAddr, int regPort, InetAddress ctServiceAddr, int ctServicePort){
        byte[] buffer = new byte[8];
        try{
            //connect to SE2CT
            InetSocketAddress regPos = new InetSocketAddress(regAddr, regPort);
            DatagramPacket packet = new DatagramPacket(buffer, buffer.length, regPos);
            DatagramSocket socket = new DatagramSocket();
            socket.connect(regPos);

            //data of registration information
            buffer[0] = (byte)(seqID/256);
            buffer[1] = (byte)(seqID % 256);
            byte[] lad = ctServiceAddr.getAddress();
            System.arraycopy(lad, 0, buffer, 2, 4);
            buffer[6] = (byte)(ctServicePort / 256); buffer[7] = (byte)(ctServicePort % 256);

            //send registration request
            socket.send(packet);
            System.out.println("Registration of the Recommender: (" + (InetSocketAddress)socket.getLocalSocketAddress()
            + " -> " + regPos + ")");
            socket.close();
        }catch (Exception e){
            System.out.println("Registration Error: " + e.getMessage());
        }
    }
}
```

A.4.2 Recommender Class

```
import java.util.*;
import java.net.*;

public class Recommender extends Thread {
    private final String vocabFile = "c:\\jay\\awwg\\data\\gums\\description\\ardescvocab.xml";
    private final String vocabURI = "http://ias.cs.unb.ca/~jay/ARDescVocab";
    private final String defaultFile = "c:\\jay\\awwg\\data\\gums\\description\\ardefault.xml";
    private final String defaultURI = "http://ias.cs.unb.ca/~jay/ARDefault";
    private final String defaultResource = "http://ias.cs.unb.ca/~jay/ARDefault#defaultAssociationRule";
    private final String key = "properties";
    //maximal length of the packet, protect from attack
    private final int maxPacketLength = 1024;
    //table structure for association rule
    private Vector arTables;
    //service port for handle the request
    private int servicePort;
    //a socket for receiving and sending packet
    private DatagramSocket sck;

    public Recommender(int servicePort) throws Exception{
        super("Start Recommender...");
        this.servicePort = servicePort;
        init();
    }

    public void init() throws Exception{
        /** initializatin for the association miner **/

        Interpreter interpreter = new Interpreter(vocabFile, vocabURI, defaultFile,
                                  defaultURI, defaultResource, key);
        interpreter.readRDF();

        //create tables for user model in the database
        arTables = interpreter.convert();
        interpreter.printTable(arTables);
        interpreter.createTables(arTables);

        //insert a default user profile
        interpreter.insertDefault(arTables);
    }

    //close socket
    public void stopThread() {
        sck.close();
    }

    public void run() {
        System.out.println("Recommender is spawned.");

        try {
            sck = new DatagramSocket(servicePort);
            while (true) {
                //get the accepted packet
                byte[] buffer = new byte[maxPacketLength];
                //create a datagrampacket object to receive requests
                DatagramPacket packet = new DatagramPacket(buffer, maxPacketLength);
                sck.receive(packet);
                //get attached message from the packet
                byte[] data = packet.getData();

                //launch a new reply thread to handle the request and generate reponse
                Reply r = new Reply((InetSocketAddress) (packet.getSocketAddress()),
                                data, packet.getLength(), miner);
                r.start();
            }
        } catch (Exception e) {
            System.out.println("Warning (Recommender): " + e.getMessage());
        }
        if (!sck.isClosed()) {
            sck.close();
        }
    }
}
```

A.4.3 Reply Class

```
import java.util.*;
import java.net.*;

public class Reply extends Thread {
```

```
//set time out
private final int timeOut = 10*1000;
//IP socket address to send back the response packet
private InetSocketAddress interPos;
private byte[] data;
//a socket for sending the response packet
private DatagramSocket sck;
private int id, dataLen;
private AssociationMiner miner;

//a constructor to create a reply object
public Reply(InetSocketAddress interPos, byte[] rdata, int rdataLen, AssociationMiner miner) {
    super("Reply");
    this.interPos = interPos;
    this.data = rdata;
    this.dataLen = rdataLen;
    this.miner = miner;
}

//stop replying
public void stopReply() {
    sck.close();
}

public void run() {
    try {
        //get the current system time
        //to keep track of the processing time for each request
        long time = (new Date()).getTime();

        System.out.print ("length: " + dataLen + "   ");
        System.out.print("Package received: ");
        for(int i = 0; i < dataLen; i++)
            System.out.print(data[i] + " ");

        //connect to the remote socket address
        sck = new DatagramSocket();
        sck.connect(interPos);

        //set time out
        sck.setSoTimeout(timeOut);

        //get the reply content
        byte[] rdata = replyHandler(data, dataLen);

        //create a datagram packet for sending packet
        DatagramPacket packet = new DatagramPacket(rdata, rdata.length, interPos);
        sck.send(packet);
        sck.disconnect();

        System.out.println("time for: " + ((new Date()).getTime()-time));

        if (!sck.isClosed())
            sck.close();
    } catch (Exception e) {
        System.out.println("Warning(Recommender Reply): " + e.getMessage());
    }
    System.out.println("Reply operator " + id + " terminated.");
}

//a method handles errors.
public byte[] errorHandler(byte[] data, int dataLen, String error)
    throws Exception {
    byte[] response = response = new byte[2 + error.length()];
    response[0] = data[0];
    response[1] = data[1];
    for (int i = 0; i < error.length(); i++) {
        response[i + 2] = (byte) (error.charAt(i));
    }
    System.out.println(error);
    return response;
}

//a method of handling all the request and generate responses
public byte[] replyHandler(byte[] rdata, int rdataLen) throws Exception {
    byte[] response = null;
    if (rdataLen < 4) {
        //print the error message if the length of the received packet data is less than 4
        String error = "message length should be longer 3";
        response = errorHandler(rdata, rdataLen, error);
    } else {
        int reqID = (int) rdata[2];

        //convert the data to a string
        String parameters = "";
        for (int i = 3; i < rdataLen; i++) {
```

```
                parameters = parameters + (char) rdata[i];
            }

            //print out the received message
            System.out.println(reqID + " " + parameters);

            if (reqID == 20) {
                //recommend associated news to the user
                Vector knownPair = new Vector();
                StringTokenizer token = new StringTokenizer(parameters, "#");
                String userID = "";
                int visitingNews = -1;
                int num = 0;
                String value = "";
                String tokenValue = "";
                if(token.hasMoreTokens()){
                    tokenValue = token.nextToken();
                    if(!tokenValue.equals("")){
                        userID = tokenValue;
                        if(token.hasMoreTokens()){
                            tokenValue = token.nextToken();
                            if(!tokenValue.equals("")){
                                visitingNews = Integer.parseInt(tokenValue);
                                if(token.hasMoreTokens()){
                                    tokenValue = token.nextToken();
                                    if(!tokenValue.equals("")){
                                        num = Integer.parseInt(tokenValue);
                                    }else{
                                        value = "num is invalid!";
                                    }
                                }else{
                                    value = "no number received!";
                                }
                            }else{
                                value = "current reading news ID is invalid!";
                            }
                        }else{
                            value = "no ID of current reading news received!";
                        }
                    }else{
                        value = "User ID is invalid!";
                    }
                }else{
                    value = "no user ID received!";
                }

                System.out.println(userID);
                System.out.println(visitingNews);
                System.out.println(num);
                value = miner.recommend(userID, visitingNews, num);

                //first two fields are exact same as received
                response = new byte[2 + value.length()];
                response[0] = rdata[0];
                response[1] = rdata[1];

                for (int i = 0; i < value.length(); i++) {
                    response[i + 2] = (byte) (value.charAt(i));
                }
            } else {
                String error = "invalid reqID for the association miner";
                response = errorHandler(rdata, rdataLen, error);
            }

            String result = "reply is: ";
            for (int i = 2; i < response.length; i++) {
                result = result + (char) response[i];
            }
            System.out.println(result);
        }
        return response;
    }
}
```

VITA

Candidate's Full Name:

Jie Zhang

Universities Attended:

Nanjing University of Aeronautics and Astronautics, 1996-2000.

University of New Brunswick, 2001-2003, Bachelor of Computer Science First Class Honours, Major of Software Systems.

University of New Brunswick, 2003-2005, Masters of Computer Science

Publications:

Jie Zhang and Ali A. Ghorbani, "Value-Centric Trust Model with Improved Familiarity Measurement", the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05), 2005.

Mehran Nadjarbashi-Noghani, Jie Zhang, Hossein Sadat and Ali A. Ghorbani, "PENS: A Personalized Electronic News System", the Conference on Communication Networks & Services Research(CNSR 2005), 2005 Halifax, Canada.

M. Kilfoil, A. Ghorbani, W. Xing, Z. Lei, J. Lu, J. Zhang and X. Xu , "Toward an Adaptive Web: The State of the Art and Science", Proc. of the Conference on Communication Networks & Services Research (CNSR 2003), pp. 119-130, May 15-16, 2003 Moncton, Canada.

Conference Presentations:

Jie Zhang and Ali A. Ghorbani, "Familiarity and Trust: Measuring Familiarity with a Web Site", Proceedings of the Conference on Privacy, Security and Trust (PST'04), October 13-15, 2004 Fredericton, Canada.

Jie Zhang and Ali A. Ghorbani, "The Reconstruction of User Sessions from a Server Log Using Improved Time-oriented Heuristics", Proc. of the Conference on Communication Networks & Services Research (CNSR 2004), May 19-21, 2004 Fredericton, Canada.

Jie Zhang and Ali A. Ghorbani,"A Generic User Modeling Server for Adaptive Web Systems", MITACS 5th Annual Conference (poster), June 9-12, 2004, Halifax, Canada.