University of Niš

Faculty of Electronic Engineering

Ana Stojanovic

I n t e g r a t i n g p e r s o n a l i z a t i o n aspec t s i n t HE p o r t a l f r a m e w o r k

Master thesis

Niš, 2002

I would like to thank

Kai Jauslin

and prof.dr. Slobodanka Djordevic-Kajan

1 Introduction

1.1 The actual situation

The amount of data available on the Internet constantly grows. That data is useful information only if it is available to the right person at the right moment. And usually that is not the case. Recent studies show that average professionals spend nearly 30% of their work time searching for information. Problem remains even if the information quest is narrowed to specific area.

When browsing university websites, for example, one will recognize deep hierarchical structure but usually with overlapping names for different things; tons of data duplicated, inconsistent and well hidden, doomed to be found when not needed anymore; document locations frequently changed but no redirection included, leaving users with "404 File Not Found" message, although the URI is correctly spelled or link from same site followed. There is one more thing not to be forgotten: *Different people have different interests*. Not everyone is searching for same piece of information when accessing the same website. It is often stressed that a user is the only entity on the world who knows what exactly he or she needs during information quest. The only real way to get individualized interaction between the user and a website is to present the user with a variety of options and let the user choose what is of his (her) interest at that specific time. [NIEL98]. Fact is that quite often, while browsing, the user does not know what s/he is looking for and great choice only confuses. The user needs help to cope with his or her inability to maintain *the underlying connections* between information sources [KOUL99].

Contemporary web-commerce applications, very often as one type of help provide recommendation. When recommending, user's view of the company's offerings is personalized, in a way that he is offered products that customers with similar tastes bought earlier. The motivation of the company owner to recommend is clear, since by recommending the right products (the ones to be bought by the customer) he increases his profit. On the universities, the situation is different: buyers do not exist and the profit that the university has from recommendation is not at all obvious. Future set the challenges for the existing research, science, education and entire environment in which academic services are provided. After long relatively slow development of university infrastructure, advance of reliable, fast communication channels of almost unlimited range, enables rapid development of physical - virtual environments [ETH00]. These virtual environments should retain original institution spirit. Therefore, along with the primary goal to make right information available to the with the primary goal to make right information available to the right people, connecting people among themselves and virtual community formation in the study groups, campus or around the whole planet.

1.2 ETH World Project

ETH World [ETHW00] is strategic initiative to prepare ***ETH Zürich*** for new information age. The goal of this project is to create universal virtual platform for communication and collaboration, which will support activities of all people who work or study ETH. ***ETH World*** will expand existing physical locations, "Zentrum" and "Hönggerberg", with virtual space, creating third virtual campus inside of Swiss Federal Institute of Technology in Zürich (***ETH Zürich***).

Traditionally, universities and other educational institutions perform research and other educational activities in physical buildings. These buildings grew, over time from representative, castle-like structures into functional constructions with hi-tech installations. Maintenance costs for this infrastructure are a part of total expenses needed for university functioning and they increase rapidly. Specialization of research rooms, flexibility of total space decreases, which again increases the need for space, and so on. With specific research group isolation their ability to adapt for new research and educational tasks decreases. The logical consequence is that universities with less physical space go ahead in development; they use greater part of their budget to improve quality of education and research. Therefore it is necessary, to find balance and develop equally physical and information architecture.

Project ***ETH World*** will try to solve this problem decreasing the need for space, with efficient use of the existing space as well with using available information technology. The virtual space will be built on the top of the exiting physical space. This project will create new type of information and intellectual environment that will support interdisciplinary research and make the university stronger as a whole. In this environment the research groups, teaching and student communities will be able to collaborate without any space or time frontiers. Opposed to other "virtual university" initiatives ***ETH World*** does not only provide distance learning ability and distributed research projects; it primarily has a goal to support and improve everyday university activities.

Shortly, aims and goals of this project are [ETH00]:

- ***ETH World*** should be understood as an instrument to improve existing and to promote new methods of research and education, without being an open university in the classical sense of distant learning. A significant expansion in the teaching and learning culture will lead to an improvement in the relationship between students and staff (learning teams), thus eliminating unnecessary hierarchical structures. Intellectual discourse between all members of the ETH community will be intensified and lifelong learning and collaboration will be fostered.

- ***ETH World*** should act as a community-forming entity. Identification with the academic institution needs to be fostered. Its users - students, teachers, researchers, staff members, alumni, and associated individuals-form a collective.
- ***ETH World*** is a network for communication and interaction. It enhances the human-machine and human-machine-human interface. Its organizational structure and the quality of its visual appearance will play a substantial role in providing accessibility and ease of movement to prospective users.
- ***ETH World*** could be viewed as an assemblage of heterogeneous components. Whereas its structure might be that of a loose assembly, essentially decentralized in its overall organization, hierarchies will, as necessary, be introduced locally. It should be conceived as a dynamic system in a state of perpetual evolution. Users will contribute actively to the growth and transformation of the system. This framework needs to be open and adaptable in its structure.
- ***ETH World***, while primarily belonging to the realm of virtual reality, must closely interact with the physical reality of existing and future facilities. The relationship between the virtual and the physical spaces must be addressed in its structure and formal manifestations. The virtual reality is to be set in relation to the physical presence of the ETH Zurich, and both are intended to develop a common identity. For that, also the physical reality (buildings, visual appearance, and infrastructure) must take account of this new identity which is made possible by the virtual campus.

ETH World conceptual solution is user oriented solution in which technology plays primarily serving but also an intelligent role.

In order to demonstrate potentials of new scientific, educational and research environment, ***ETH World*** started pioneer projects in various areas, among which are research tools, *e-learning*, community building, external relations, information management and tools for infrastructure. Database Research Group project "Information Search and Coordination", whose part is my master thesis, is related to information management.

There are two big problems in contemporary information systems: how to achieve consistency of information, originating from different multimedia sources and how to support user-oriented and effective search mechanisms for information discovery in enormous, virtually integrated data space. In order to help, system has to know much more about the user than he is ready to reveal. What makes it even harder is well-known phenomenon in user interface design ("paradox of the active user") that people are more motivated to start using things than to take the initial time to learn more about them or set up a lot of parameters [NIEL98].

1.3 Personalization

Personalized system should gain users trust by building a meaningful one-to-one relationship; it has to understand the needs of each individual and help him or her satisfy a goal. That goal efficiently and knowledgeable addresses each individual's need in a given context. One of the elements of personalization is building a balanced relationship with the user.

The other two elements of personalization are *customization* and *recommendation*. Customization allows users to build their own user interface by selecting from different information channels. Recommendation techniques represent system ability to use behavior and group preferences or current task-specification to recommend interesting items to an active user.

Information architecture in this system includes components coming from three different domains: working context, information contents and users. Users have profiles, which represent their interests and their behavior. Document contents have certain features which influence personalization (document author, location, rating, topic…). Working context provides rules that determine the way the personalization happens [INST00].

Let us go back to the university example. In this environment there are certain limitations that ease up the personalization process:

- Potential user groups are easy identifiable. This fact imposes collaboration: *Students* are usually interested in information on their lectures, exams, literature or some information on special events; *academic staff* in research and publications, *potential students* in more information on how their life is going to look like if they begin studying at that university. *Administrative staff* should keep everything consistent and contemporary. Each of these groups has specific view of (entrance point into) university still very often these views overlap.
- Rules are strict and well known, integrated into university constitution, statute, and schedules.
- There is a lot of information available but not used. Administrative data on students can be used to provide them better view of what they are able and what are not allowed to do.
- Information is area specific and usually very similarly represented (this implies that some standards can be adopted).

Although people do not like to be stereotyped, university is the scene where lots of differences are not visible because interests of groups of users often partially or even totally overlap. The latter facilitates the use of collaborative filtering.

With these in mind, knowing the goals of the ***ETH World*** project and the feature of a typical portal to provide the user right information at the right time, together with the search capability, the solution imposed is the portal itself.

And while the organization and communication primitives of the ***ETH World*** project are defined and published by the winner team of the conceptual solution tender, many different technical aspects should be solved with experimental studies and the implementation of the ***ETH World*** project. One of the solutions is the portal framework that was developed, as a prototype, by the Database Research Group at the Swiss Federal Institute of Technology in Zürich.

1.4 Portal

„Portal is an exiting, interesting and focused view to one part of the Internet – be it past, business sector, one organization, company or view to on-line world" [PRAC].

„Portal is an application or a device that provides users a personalized and adaptive interface for people to discover, track and interact with other relevant people, applications and content [MOR99]".

There are many other portal definitions (see chapter 2), since the word ***portal*** appeared in the on-line dictionary all at once, was accepted from various web site models, can be used for almost anything and exists in several different variations - vortal, personal portal...

ETH World portal should be an entry point into the virtual environment customized in a way that corresponds to a specific user needs. In this way portal can be understood as a central interface between infrastructure and that specific user. Infrastructure is consisted of services (applications) that are offered to that user by the means of the portal. Portal actually plays double role: for applications it is a run-time environment (*framework*), while for a user it is an information presenter. Portal should, as interface, fulfill several requirements, and some of them are:

- *Dynamic presentation.* Selection and presentation of corresponding personalized information on different terminals. This is achieved by using dynamic, not static structures, for variable content pages representation. This concept supports total separation of presentation and information. It is required to separate programming logic and specific user data as much as possible. This way site maintenance becomes easy and any data change simple.

- *Personalization*. Individual entry point and authentication i.e. one user identification (*single sign-on*). Information on specific user is transparent for different applications, which enables them to adapt according to a specific user needs.
- *Framework*. A large number of services and libraries that provide simple application deployment. The integration of external information sources should be provided with minimal additional costs.
- *Maintenance and Deployment*. Portal provides authors with clear interface offering them corresponding programming support.
- *Security*. Interface provides user identification, authentication and authorization.
- *Scalability*. Concept should, basically, work with large number of users and under heavy load.

ETH World portal prototype enables user and session identification and dynamic presentation creation, in a two step process:

- application call and output transformation, and
- Adaptation of certain application dependent user parameters (*customization*). These parameters are, at the same time, available to template as XSLT parameters.

1.5 Technology

System is implemented using JSP [JSP] and Java Servlet technology [JSWP]; XML is used for data interchange between different applications, as well as between certain applications and existing framework. XSLT is used for transformation of XML documents into different presentation formats (HTML, SVG, PDF, WML etc...) in accordance to specific user needs and used input-output device.

JSP technology is used because it inherits all good features of Java technology, including platform and server independence, modular architecture with reusable components, and access to all Java APIs (like ***JDBC***, ***JavaMail,*** and ***Java Transaction Service***). The feature that makes JSP technology unique is ***Custom JSP Tags***. It refers to creation of *custom tag libraries* of frequently used functions and their distribution to large number of authors (***Java Community Process***). This way is achieved the total separation of *look and feel* and logic in every created page. Since the tags are executed on the server side, on client side they do not impose any further limitations. Additionally, since each of the tags implements specific functionality their use becomes intuitive and therefore understandable for non-programmers. Tags enable generation of new and manipulation of already existing information; for example, the use of JDBC enables application programmers access data that resides in already existing databases; the use of Java-COM Bridge enables communication with Microsoft components, that are attached to the portal framework; they provide access and generation of XML documents, different kinds of filtering…

The aim of the prototype deployment was to use as much as possible *off-the-shelf* components. Since the goal of the project as a whole was to prove that the use of *collaborative filtering* techniques in academic environment can encourage and improve real-world communication and formation of virtual, as well as real communities (ref. section 1.6), and not discovering a new one, the ***Microsoft Analysis Manager*** is used for entire user population segmentation. For communication with this tool OLE DB for Data Mining is utilized, since it is supported from other *data mining* tools, which enhances portability.

1.6 The Goals

The goal of this master thesis was the extension of existing portal framework with module that enables recommendation. Since the basic functionality of every portal is to present the user the information that he will either find interesting or need at that specific moment, one way to do it is provide him/her a view to all the topics that were seen by the people with similar interests. Since this means taking out all the users that have no influence on that specific topic and all the information of no significance for that topic, this process is also called *collaborative filtering* (CF). The term *collaborative*, in this context, is defined in [GOLD92] and includes the situation in which people help each other (i.e. collaborate) by filtering unimportant information. They actively react on presented information in a way that they, for example, mark read document as interesting or particularly uninteresting. The idea is not at all new (ref. chapter 3) and is already used in many *web-commerce* systems, one of which is, for example ***Amazon.com*** (where user's interest in a specific product is indicated by his/her buying it). Explicit identification of specific users in a virtual community, additionally, encourages *real-world* communication and building social networks. This feature, though of absolutely no importance for *e-commerce* environment becomes valuable in an academic environment.

The developed recommendation module (***advisor***) is a part of the framework (ref section 5.2) and its function is to help user in selection process. Identified user is in every selection process recommended a list of options that the system *thinks* are the most interesting to him. When using this module, application programmer, needs only to provide the criteria upon which the recommendation is generated. The ***advisor*** module is developed as a JSP tag library, which makes it (but not totally) independent of portal framework. Currently this tag library provides use of *data mining* clustering technique that enables segmentation of entire user (or information) population according to the imposed criteria. This way new information on grouped subjects is discovered. Newly discovered information can be utilized to help system users during their interaction with the system and make their experience more pleasant and useful. The presumption that stood during this module deployment was that were no *security* or *privacy* problems i.e. that the system is available all the information on all the users and that it can utilize them for knowledge discovery.

Additionally, some applications were deployed (for example timetable, course selection application…- ref chapter 5), that will provide collection of new data on individuals and user groups. These applications provide the user with information, whose structure and content are profiled according to users needs and respect the rules imposed by the environment. The applications are developed according to the specification for application development in the portal framework [JAUS01] and utilize all the advantages of the JSP/XML technology that underlies this specification.

The proposed research was realized on Institute for Information Systems ETH Zürich and personalization module is integrated into existing ***ETHWorld*** portal prototype.

2 Portals

2.1 Portal is...

Great and impressive entrance (frequently used in metaphors); for example, "cathedral portals", "portals of Eden", "portals of success".

And in the Internet dictionary:

Portal is an application or device that provides a personalized and adaptive interface for people to discover, track and interact with other relevant people, applications, and content [MOR99].

Portal is an application that collects contents relevant for the actual user [CHU00].

Portal is an exciting, interesting, and focused view to the specific part of the Internet – be it past, business sector, one organization, company or a view to an on-line world [PRAC].

Portal provides secure, single point of access to various information, business processes, and people, personalized in accordance to users' needs and responsibilities [IBM01].

Portal is a gateway or an entrance to the Internet; homepage in a Web browser, for example ***Infoseek, Excite, Yahoo, Lycos***, and so on. Generally, portal is a web site that offers a broad range of resources and services, including e-mail, discussion forums, search capability or some other on-line services. It is deployed as a response to user needs to filter information and receive fast and simple information access to web whenever they need it [MIL01].

Portal is a technology that enables one company or organization to open itself internally and towards the world, and offer its users unique gateway to personalized information, necessary for making important decisions. Portal is an amalgamation of software applications that arrange, manage, analyze and distribute information across and outside the organization (and include *Business Intelligence,* *Content Management, Data Warehouse,* and *Data Management* applications). Portal, mainly, presents an ability to modernize information access.

Portal is a personalized gateway that combines all relevant information sources into unique user's world. Portals enable users to track their business tasks, receive information arranged according to their taste and communicate with their colleagues. Portal should seamlessly connect all Internet and organization intranet elements and provide users ability to search both, Internet contents and internal documents.

Word ***portal*** means many different things to many different people. Uncertainty is built into its history, since it suddenly appeared in the on-line dictionary, was accepted for several different web site models, can be used for almost anything and already exists in several different variations - vortal, personal portal, corporate portal...

Simply put, portal provides right information to right users at the right time using web, mobile and other devices. Mobile devices and *pervasive computing* concept discover the other side of the portal. It is possible to get all the necessary information on some organization by accessing its web site or using any Internet portal, any time and from any location. Information is adapted to specific user needs independent of the way it is accessed, through PC or some mobile device. Using aggregation and personalization (figure 2.1), portals provide every user with an individual view on the organization web site or individual entrance to Internet and provide them efficient access to all the necessary information and application. For *pervasive computing* devices this ability is critical, since the network range and available resources are extremely expensive. *Wireless* portals provide user with time and location dependent and user specific content.

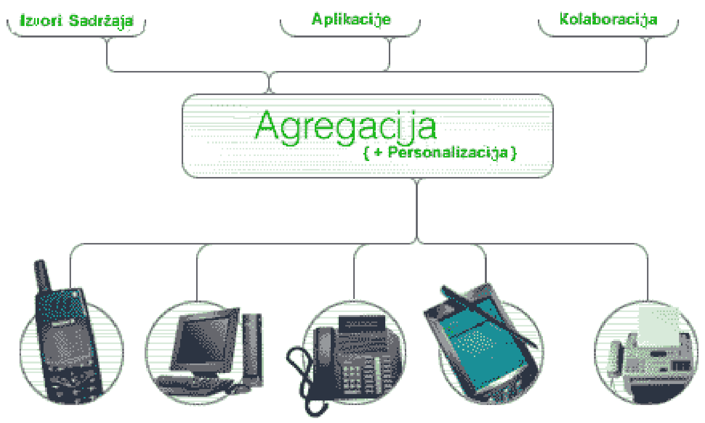


Figure 2.1 Basic portal functions

2.2 vs. Home page

Portals, more and more, replace *home pages* of different organizations. Integration of different services and personalization differ portal from owner-oriented home pages. Portals are not horizontal entrances to a web site, but an exit to the market of various services.

The basic distinctions between home pages and portals are [GNAG01]:

- *Home page is defined with static contents, and portal with dynamic.* In a certain way traditional *home page* is like a publication: Once written it is distributed to different users, which read it. In that sense portal is similar to a publishing company. Portal continuously collects information from various information sources, for example university databases and external news feeds and hands them to the correspondent public.
- *Home page provides links to information; portal connects the user directly to information tools.* Though this may seem like a small difference, it is not. "Old-fashioned" web site can provide link that leads to a library home page, that demands a log to a catalogue ...while one logging to a portal, provides a user direct service utilization, for example catalogue or e-mail or course monitoring – without need for any further logging.
- *Home page is the same for every user; portal is different for different users.* Portals aim to provide adaptive home pages (for example, ***Yahoo*** enables adding and deleting different information sources and links). In an university environment this kind of site should automatically recognize if the user is a student, lecturer, potential student or alumni—and offer him adapted information, services and data access.
- *-Main purpose of a home page is information; while main purpose of a portal is business.* When constructing a *home page,* various facts, plans, reviews and editions are collected. Portal contains all these but its basic function is to collect something more: a heap of e-mail and search programs, personalized calendars, selection tools, course access and manipulation and so on. While a *home page,* mainly collects, formats and presents information portal provides additional functional layers: it controls credentials and access rights, passwords, and integrates databases and collects payments.

2.3 Portal types

2.3.1 Focus

Portals are, usually, focused on some subject - interests, business sector or organization (external and internal portals). In any case, main benefit from using portal approach is saving user's time by taking out large part of "noise" from

Internet, and identification of interesting sites and contents. According to the range of themes they cover, two different types of portals can be distinguished:

- *Horizontal portals* (general-purpose portals – megaportals). These portals list contents available on Internet, act as *hubs* that help users locate contents needed and access them. Business model of this type of portal is based on selling space for advertisement *banners* and successful directing web surfers to the destinations wanted. In this way they insure their existence. At the moment, these portals offer access to a broad specter of on-line information sources and services. Although there is no unique model that defines portal, all portals have at least next four basic features: they enable web search, show news, and provide tools for referencing and some communication capability (for example, e-mail or chat). The most famous are ***Yahoo, Excite, and AOL***…

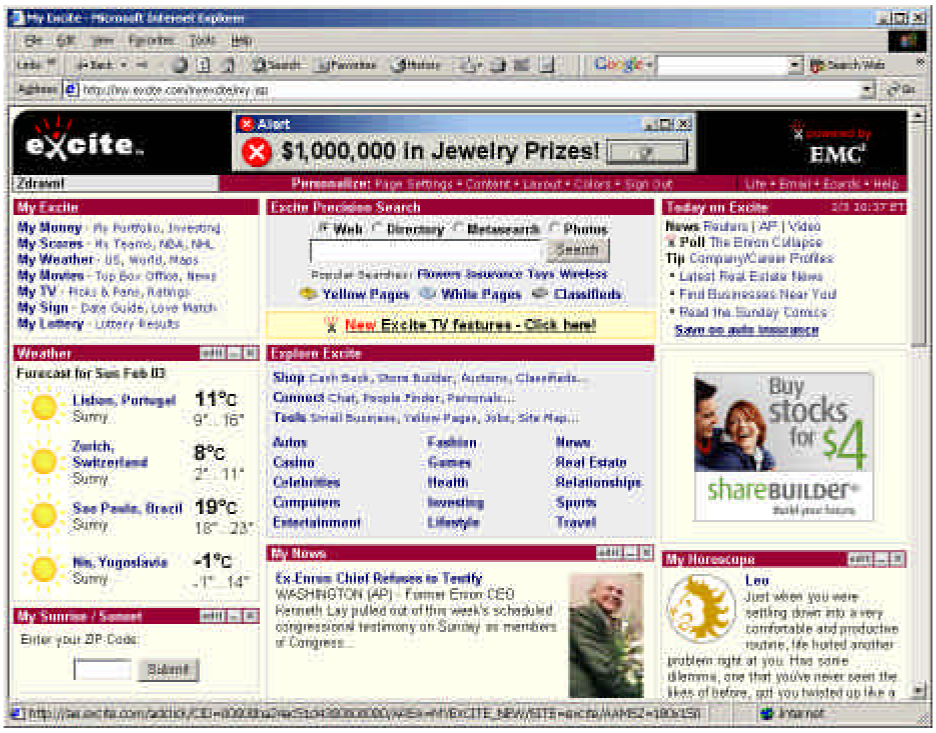

excite
Alert
$1,000,000 in Jewelry Prizes!
powered by EMC
Zdravo!
Personalize: Page Settings • Content • Layout • Colors • Sign Out
Life • Email • Boards • Help
My Excite
My Money - My Portfolio, Investing
My Scores - My Teams, NBA, NHL
My Weather - US, World, Maps
My Movies - Top Box Office, News
My TV - Picks & Pans, Ratings
My Sign - Date Guide, Love Match
My Lottery - Lottery Results
Excite Precision Search
Web Directory Metasearch Photos
Search
Popular Searches: Flowers Insurance Toys Wireless
Yellow Pages White Pages Classifieds
New Excite TV features - Click here!
Today on Excite
News Reuters | AP | Video
Poll The Enron Collapse
Tip Company/Career Profiles
• Latest Real Estate News
• Find Businesses Near You!
• Read the Sunday Comics
Save on auto insurance
Weather
Forecast for Sun Feb 03
Lisbon, Portugal Sunny 11°C 9°..16°
Zurich, Switzerland Sunny 8°C 2°..11°
Sao Paulo, Brazil Sunny 19°C 18°..23°
Nis, Yugoslavia Sunny -1°C -1°..14°
Explore Excite
Shop Cash Back, Store Builder, Auctions, Classifieds...
Connect Chat, People Finder, Personals...
Tools Small Business, Yellow Pages, Jobs, Site Map...
Autos Fashion News
Casino Games Real Estate
Celebrities Health Relationships
Computers Investing Sports
Entertainment Lifestyle Travel
Buy stocks for $4
shareBUILDER
My Sunrise / Sunset
Enter your ZIP Code:
Submit
My News
Ex-Enron Chief Refuses to Testify
WASHINGTON (AP) - Former Enron CEO Kenneth Lay pulled out of this week's scheduled congressional testimony on Sunday as members of Congress...
My Horoscope
Leo
Just when you were settling down into a very comfortable and productive routine, life hurled another problem right at you. Has some dilemma, one that you've never seen the likes of before, got you twisted up like a
Internet

Figure 2.2 One horizontal portal - Excite

- *Vertical portals* (vortals): web sites that provide unlimited amount of information that refer to relative narrow areas of interest. This is currently fastest growing type of Internet portals. Vertical Internet portals can be distinguished from horizontal portals in the range of users that they target. Horizontal Internet portals try to serve the entire Internet population, while vertical portals target niche audience – usually interested in a specific activity. Vertical portals provide contents, relevant to that organization, with links to the organization, partners and even web sites of concurrent organizations. One of the goals of a vortal is to provide community and collaboration capabilities that aim this activity or group, and *e-commerce* services for products and services relevant for that activity or group. Vertical portals become more and more popular, especially in the business-to-business (B2B), business-to-customer (B2C) and business-to-employee (B2E) domains. Vertical portals should be built over the horizontal portal infrastructure.

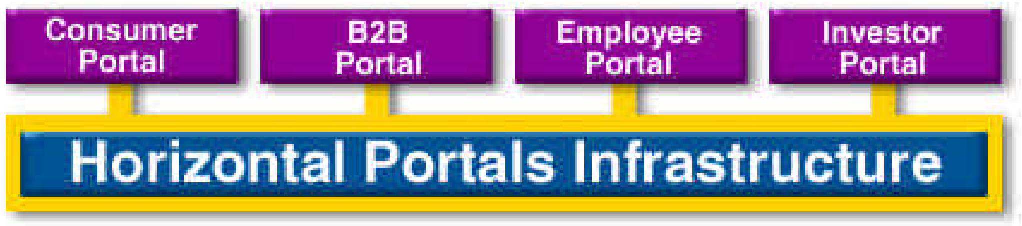
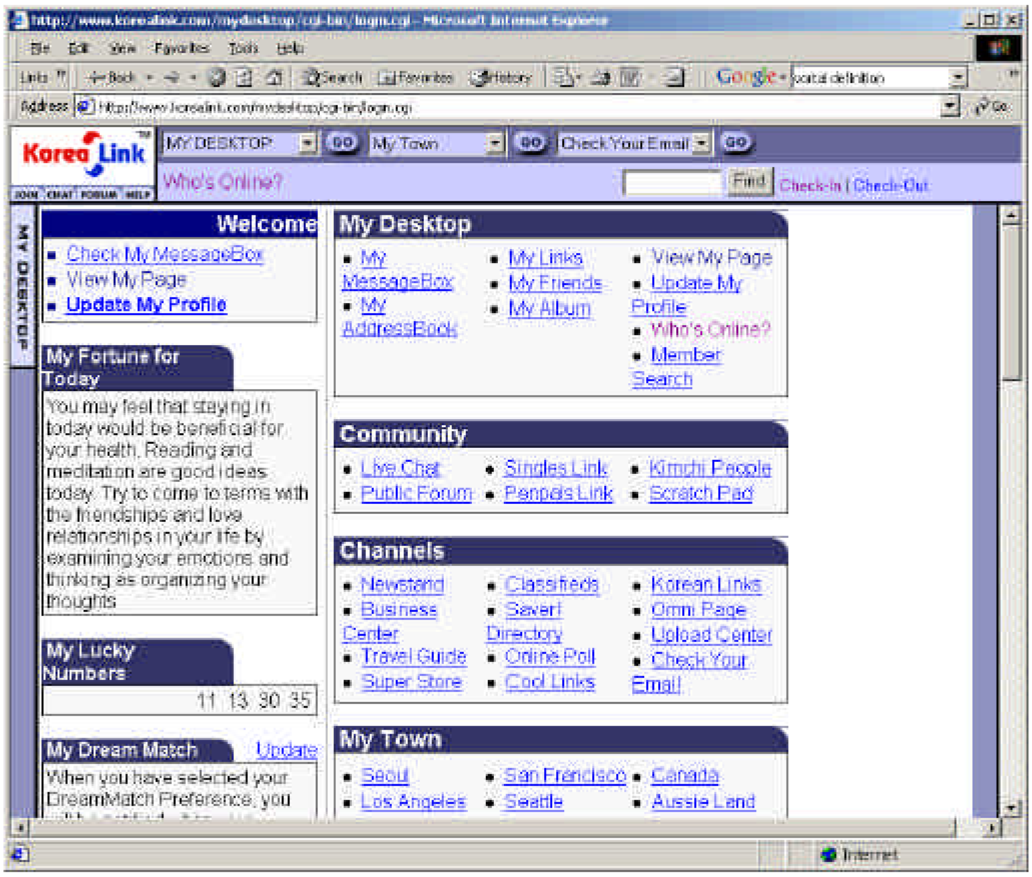


Figure 2.3 Architecture of a vertical portal

One of the key features of all the vortals is the way they inform their users about existence of the Internet resources. Main objective of the vortal is to "painlessly» direct its users to the best resources for specific domain.

- The most important tool of the vortal is *discussion forum*, location where all the equal-minded people can meet, discuss about problems from their interest domain, send messages and receive answers.
- On-line, up-to-date *calendars* inform portal users about important upcoming events. These calendars are typically global by nature, organized by subtopics and searchable by geographic regions and many other useful ways.
- *Classified advertisements* are frequently one of the most important vortal resources. There users can publish (or search) articles of common interest. This "billboards" besides search enable sending messages and searching other information available on that specific product or service.
- *Rota ting banners* are a usual part of a vortal. Good advertisements inform users about potentially valuable products and services, and if they are not too imposing, large, colorful and have right content. At the same time they cover provider costs, whose main goal is creating a site that would be relevant to its users.
- *On-line surveys* can provide valuable information for vortal users. A large number of users are to publish their opinion on some subject.
- And at last, *search engines*. Since the goal of the most of the vortals is, to totally, organize all Internet resources, that are related to the specific topic, into one site alone, most of them, at the same time, starts its own life. As the Internet grows, grows the vortal. Without good search tool a great part of portal capacity stays unused. Good search engine provides users many advanced search techniques.

Figure 2.4 shows vortal KoreaLink.com. In it everything, from the best Kim chi restaurant in USA to Korean keyboard signs can be found. Chat, singles forum and much other information on Korean community, products and services can be easily located and searched using this portal.



You may feel that staying in today would be beneficial for your health. Reading and meditation are good ideas today. Try to come to terms with the friendships and love relationships in your life by examining your emotions and thinking as organizing your thoughts

When you have selected your DreamMatch Preference, you

Figure 2.4 An example of a vortal: KoreaLink

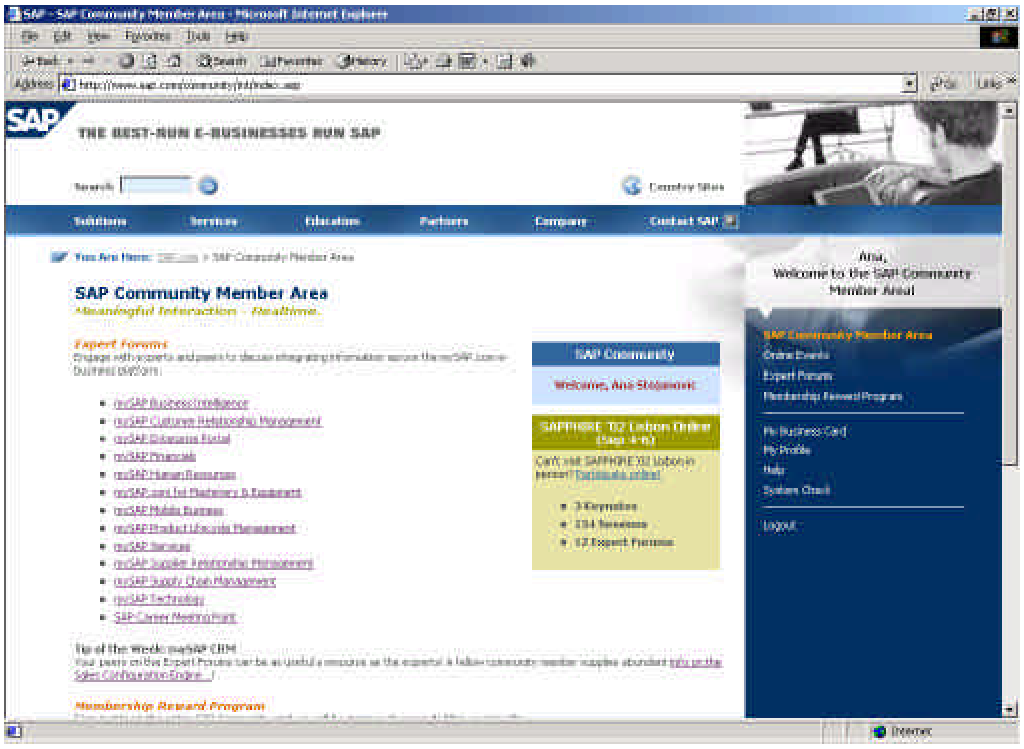
A separate subgroup of vortals consists of corporate portals.

2.3.2 Corporate Portals

Corporate portals connect users, not only to all the information required, but also with all the people necessary and provide the collaboration tools. This means that corporate portal provides access to all the *groupware*, *e-mail*, *workflow* and *desktop*, even critical business applications.

There are at least four sub-types of corporate portals:

- *Information portals* connect people with information, by organizing large collections of content based on subjects or themes.
- *Collaboration portals* enable teams of users to establish areas or working groups for virtual projects along with tools for collaboration. This way they provide them with the possibility to collaborate.
- *Expert portals* connect people together based on their skills and experiences, as well as their information needs.
- *Knowledge portals* are the combination of all of the above to deliver personalized content, based on what each user is actually doing.


SAP - SAP Community Member Area - Microsoft Internet Explorer

Address http://www.sap.com/community/int/index.asp

THE BEST-RUN E-BUSINESSES RUN SAP

Search

Country Sites

Solutions · Services · Education · Partners · Company · Contact SAP

You Are Here: [illegible] > SAP Community Member Area

SAP Community Member Area

Meaningful Interaction - Realtime.

Expert Forums

Engage with experts and peers to discuss integrating information across the mySAP.com e-business platform:

- mySAP Business Intelligence
- mySAP Customer Relationship Management
- mySAP Enterprise Portal
- mySAP Financials
- mySAP Human Resources
- mySAP.com for Machinery & Equipment
- mySAP Mobile Business
- mySAP Product Lifecycle Management
- mySAP Services
- mySAP Supplier Relationship Management
- mySAP Supply Chain Management
- mySAP Technology
- SAP Career Meeting Point

Tip of the Week: mySAP CRM

Your peers on the Expert Forums can be as useful a resource as the experts! A fellow community member supplies abundant info on the Sales Configuration Engine...

Membership Reward Program

SAP Community

Welcome, Ana Stojanovic

SAPPHIRE '02 Lisbon Online (Sept 4-6)

Can't visit SAPPHIRE '02 Lisbon in person? Participate online!

- 3 Keynotes
- 234 Sessions
- 12 Expert Forums

Ana,
Welcome to the SAP Community Member Area!

SAP Community Member Area

Online Events

Expert Forums

Membership Reward Program

My Business Card

My Profile

Help

System Check

Logout

Figure 2.5 An example of a corporate portal (SAP community portal)

Concerning the information access corporate portals differ from Internet portals in that they are:

- *Comprehensive*: Unlike web portals they provide access to a lot of various data formats.
- *Organized*: they organize access to information so that users can browse them.
- *Personalized*: they collect individual views to relevant information and notify users on the availability of new material via electronic mail or other media.
- *Location-transparent*: They organize data access but they do not store the data itself.
- *Extensible*: They support system extensions in a way that they provide the capability of cataloging new types of information.
- *Automated*: They automatically identify and organize access to new content.
- *Secure*: They provide selective broker access to internal corporate information.

2.4 What makes a portal a portal

Portal is an application or a device, which provides a personalized and adaptive interface for people to discover, track, and interact with other relevant people, applications, and content. Key features of a portal are [MOR99]:

*P*ersonalization for end user is the most critical features. A portal must deliver a personal or community desktop for every user by establishing unique look, content, and application interfaces and *proactively* rendering them based on the user's role in the community or by actively tracking the user's individual usage, interests and behaviors.

*O*rganization of the users working space to eliminate the information glut. Users want consolidated access to relevant people, applications and contents. The concept of *stovepipe* applications became a thing of a past. Organization wants simple controls to design their desktop in a way that suits them.

*R*esource division determines who sees what. Portals must have membership services layer for user authentication, *single-sign-on* and credential mapping. Users demand the highest level of security, but the least amount of annoyance.

*T*racking of activity provides users with a payback for using the portal. The more users use the portal, the more it becomes tailored to specific interests and affinities the user may develop. Though, this may sound threatening at first, users should have the ultimate control over what gets tracked.

*A*ccess and display of aggregated multiple heterogeneous data stores, including relational databases, multidimensional databases, document management systems, e-mail systems, web servers, news feeds and various file systems/servers (audio, video, image, and so on). It is extremely helpful for users to see their e-mail next to news feeds, and a list of on-line users who can help them understand the information while maintaining a single context.

*L*ocation of important people and things. A portal is based on the basic desire of users to easily find information and people by searching or navigation. The means that can help users passively (or actively) discover the experts, communities and content in a relevant context, should exist.

2.5 The Basic Portal Components

Portal is a tool for managing intellectual assets of an organization, its content and data. Basically, business information portals, *business-to-business*, *business-to-employee*, university and public web portals all have to fulfill the same requirements. They all need to have scalable infrastructure, flexible and powerful presentation framework that enables simple portal building. All of them demand high level of personalization so that the user is delivered the most valuable information, and therefore enable more useful interaction and intensify user's loyalty to the portal. Specific types of the portal, off course, have some unique requirements. Depending on the nature and volatility of information, some kinds of a portal can demand higher level of security that includes specialized forms of authentication and access control. Depending on the number of users, some portals can demand high level of accessibility and the ability of extension. Customer portals, generally, allow users to log-in and manipulate their accounts. Opposed to it, business portals frequently require integration with existing user databases or access systems. University and other types of institutional information portals have to provide their users all those services that are available to them in the physical institution.

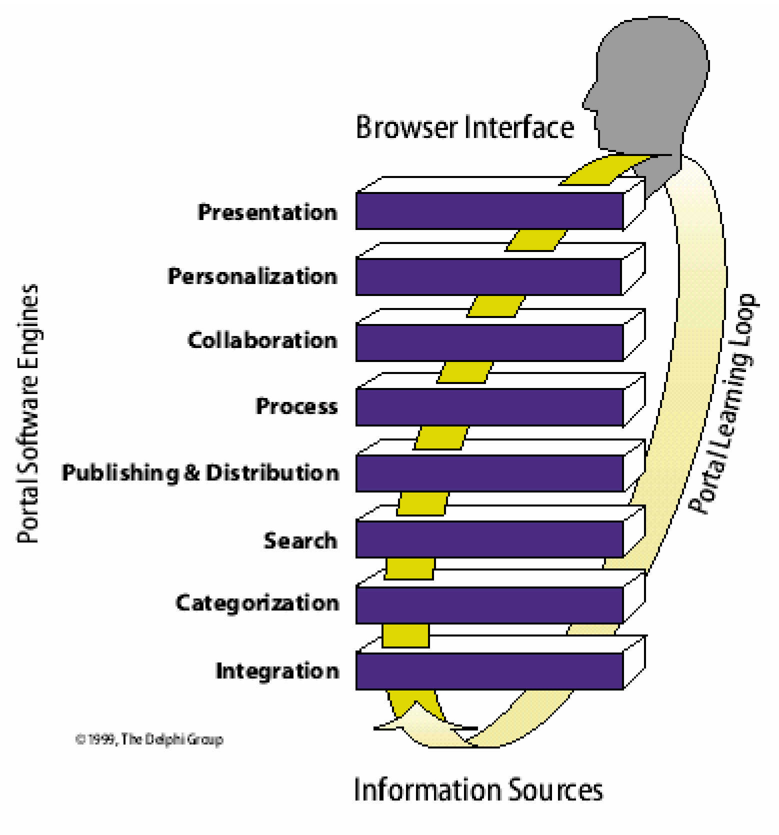


Figure 2.6 The architecture for one portal

Due to the complexity of the challenges that are posed to a portal, it requires following nine architectural elements (figure 2.6) [DG00]: *Presentation, Personalization, Collaboration, Process, Publishing and Distribution, Search, Categorization, Integration* and *Learning Loop*.

2.5.1 Presentation

Portal accesses a large number of information channels and all this needs to fit comfortably in a small display space. Besides that it has to support "zero training" operation for users. The basic features of this architecture component are:

- *Color*: color schemes establish the look&feel of a portal and can be used to more readily communicate data. Users are frequently reinforced by the ability to personalize the look of their portal and color is an easily managed attribute for most interfaces.
- *Application layout*,
- *Dynamicity*, portals can present information to their users based not only upon user role but also upon how user accesses the portal and what process the user is engaged in.
- *Device Independence*, ideally the portal will manage all user devices appropriately; but this is a tall order in today's e-crazed environment with portals accessed by everything from desktop and laptop browsers to PDAs and cell phones.

2.5.2 Personalization

Personalization is a critical portal ingredient as it provides both productivity enhancement and effective individual information management. This component filters the information specifically for the user's working style and content preferences. Personalization has become necessity because the volume of information available in the Internet-enabled electronic business environment has outstripped the capacity of the individual to organize and process it. This component offers two new value propositions to end-users:

- they can elect to display/not display particular categories or channels of content and
- They can control the placement and prominence of the content displayed.

Personalization includes: user interests and profiles, *publish subscribe* information access, storing event sequence, putting filters, customization capability, context help and preposition based learning (reference section 3.1).

2.5.3 Collaboration

Collaboration is the component, which expands portals from passive information kiosks to new discussion forums that enable interaction among any combination of users. Portals should enable:

- Asynchronous communication (for example, discussion groups or room for team members that collect all documents, plans, and everything else that belongs o that group of users), and
- Synchronous communication (for example, chat forums).

There are six types of collaboration that a portal should support: synchronous live (*chat*), asynchronous linked (e.g. questions and answers), asynchronous separate (e.g. e-mail), content development (e.g. document revision), group polling, moderated (building consensus or finding common perspectives).

2.5.4 Process

Since most portals focus not just on information accessibility but on *e-business* management, process support is critical to many portals. This is the component that enables portal users to initiate and/or participate in online *e-business* processes. Increasingly, the need to introduce process automation capabilities (such as those found in today's commercial *workflow* software) to enable predefined process flows monitoring. Portals should support the following six process types:

- *Integrated processes* – are supported by infrastructure external to the portal itself, and are integrated "transparently" into portal services.
- *Inherent processes* – are supported by infrastructure internal to the portal.
- *Wide-area support* – to support workflow and processes across wide area networks, the portal must manage a variety of communication protocols and user interfaces.
- *Role-based processes* – role-based services support standard processes based upon the common roles of the user (e.g. call-center supervisor, clerk …). Most often these processes are internal to the business.
- *Modifiable processes* – there exist a large number of organizations that support user modification of processes. Very often, these processes are internal to the business.
- *e-Process linked* - portal should enable transparent linking to existing *e-business* processes. These processes already exist in external or partner organizations.

2.5.5 Publishing and Distribution

This component supports creation, authorization, inclusion in portal content collections and distribution of structured and unstructured information, in multiple on-line or hardcopy formats. These services are usually referred to as *Content Management* services. The goal is to support creation and flow of information in the organization while minimizing the required portal infrastructure and administrative support. This architectural layer provides:

- *Author support* – this is a set of tools available for authors to create, publish and maintain the portal content
- *Posting control* – frequently information should be available on precise days and times.
- *Modification* - users must be able to significantly modify documents.
- *Conversion* - automated, transparent conversion services ease the burden on the author and administrator, making the information collections more useful and increasing utilization.
- *Internal contents* - are contents created and published in the organization that rely on the portal infrastructure.

- *External contents* – are contents that the organization syndicates or acquires from external sources, and rely on the infrastructure external to the portal. For example, *news clipping* services.
- *Link Integrity* – since the most of the portals provide information to users through hyperlinks, maintaining their integrity is of critical value.
- *Records Management,*
- *Renditions* – rendering contents in various formats
- *Derivations* - portals should ease the management of multiple versions of commonly derived objects. For instance, a 35mm high-resolution photo used for annual report might also have scanned medium-resolution derivative, useful in a corporate newsletter and a third data-light, lower quality version used on the web site.

2.5.6 Search

This layer in the portal architecture provides tools for identification and access to specific information within document collections, available at and through the portal. Many collections of information are quite large. Enabling the effective search across the information requires indices or other management structures. Portal should support different combination of the following search schemes:

- *Text box* - user types in a keyword. "Full-text" search capability is the best for the advanced users, the ones that know what they are looking for.
- *Parameterized or led* – the user is, usually, offered a set of parameters and the user marks just the ones they found important. This type of search tools is good for users that have an idea about what they are looking for and need help to refine their search criteria.
- *Case-based reasoning and suggestive search* - this type of parameter search leads user through the sequence of steps, in order to find the best choice. The system suggests the further query refinement, based on the observation of user actions during the search process. Opposed to the led search the part of information accessed this way is not chosen according to the user criteria. This system is efficient in the cases when rules can be defined or if/then analysis is possible.
- *Predefined search* – this type of search depends on site search history and discussion with users that frequently access that site.
- *Collaborative filtering* – this tool suggests categories or specific documents, which the user could find interesting. Suggestions are formed based on the selections of the users with similar profiles and/or behavior.
- *Natural language context-based* - users have the ability to present their requests in their native language, since the system will be able to understand their query in the active context.

2.5.7 Categorization

The benefit that categorization brings is providing the context. This component implements organization-specific taxonomy that helps understanding information and enables fast recognition and useful utilization of this information. The four most common categorization approaches are: automatic, manual, dynamic and modifiable. The most frequent is the categorization that combines all of them.

- *Automated categorization* relies upon software tools that assess information collections and repositories. These tools rely on statistical and semantic methods, and extract taxonomic characteristics and hierarchies.
- *Manual categorization* relies upon architects, analysts and users. Although this approach gives accurate results, which are tailored to the organization, it is labor intensive and challenging to maintain.
- *Dynamic categorization* is a type of automatic approach. Information hierarchy is dynamically maintained and updated based upon additions and deletions to the repositories and upon changes in the organizations processes and information use patterns.
- *Modifiable categorization* - very often different categorization taxonomies are required in the various departments of the organization. This is why, in the Internet era, users should have the ability to modify existing categories and adapt them to their needs.

Categorization includes: taxonomy organization (systematization), index search, automated classification collection, segmentation, metadata manipulation, and discovery of new contents, connections among structured and unstructured data, business knowledge, and ties to external contents.

2.5.8 Integration

The success of most portal implementations depends on integration - incorporating structured data (e.g. legacy systems and data warehouses) and unstructured information (everything else, from e-mails to word-processing). The wide range of potential information sources, the breadth of user requirements and the unpredictability of information needs at any given point in time present challenges that are not addresses by traditional application development and maintenance cycles. The challenge of the portal is to create the network of information sources required to responsively and flexibly support the specific day-to-day knowledge requirements of the portal's users. There are six categories of information sources: internal and external legacy systems, internal and external databases, and internal and external new data.

2.5.9 Learning Loop

This component differs from others in that it is not concerned with a specific aspect of information management, but in the ongoing effectiveness of the portal itself. This component enables the portal to adjust heuristically to changes in the organization's work and information environment. In order to be effective, portal should, from the very beginning contain a learning component. Though it may seem a bit scary, this requirement could be fulfilled by use of available and well-understood analysis tools, including usage metrics (e.g. who are the portal users, where they are searching for information etc.), content evaluation (identification of those information elements that are of value to users and those that are not) and intelligent content management (agents, neural networks, natural language processing). Learning loop utilizes the following eight services:

- *Authoring & posting* - portal should monitor users authoring/publishing habits and use them to modify the information presented.
- *Page Hits* – it is possible to assess the way the portal users access portal contents. When mapped to user profiles page hits provide a key means for monitoring and improving the effectiveness of the portal.
- *Access Habits* – the types of devices used to access portal can be analyzed to find better ways of access and improve portal support.
- *Community* – by being attentive to the user community and by getting used to their style and the information content, portals become very valuable to their users and create more enduring and fruitful relationships among members of the community. Learning loop informs users and enables the development of community and community services.
- *Transactions & processes* – supported by the portal give significant information on portal users. This information provides insight into potential improvements in productivity.
- *Process involvement* – by providing detailed information on the supported processes, learning loop enables refinement of user tasks and refocuses the activities where the most benefit can be gained. When users change their activities, learning loop recognizes these changes and adapts.
- *Preferences* – when the learning loop provides enough information on user activity, analysis can determine additional valuable services and alternative user preferences that can improve portal influence on costs, quality and cycle time. When the preferences are changed learning loop should identify the change and help user find the information needed.
- *Polling* – is a way of providing users to proactively instruct the portal on modifications they would like to see. It can be very effective means of rating already deployed contents.

2.5.10 Portal Foundation Elements

One of the portal goals is to coordinate existing services and the ones to be developed and try to improve their effectiveness or their value. Portal must complement information technology context it exists within. This is especially true for the portal foundation elements in the areas of security, directory services and authentication.

2.5.10.1 SECURITY

Since the portals potentially provide users large amounts of information, one of the most important portal tasks is to provide that only the right people can access the portal environment and only the systems they are authorized to. The basis for working with the personalized web applications are user authentication and secure *single sign-on* mechanism (ref figure 2.7).

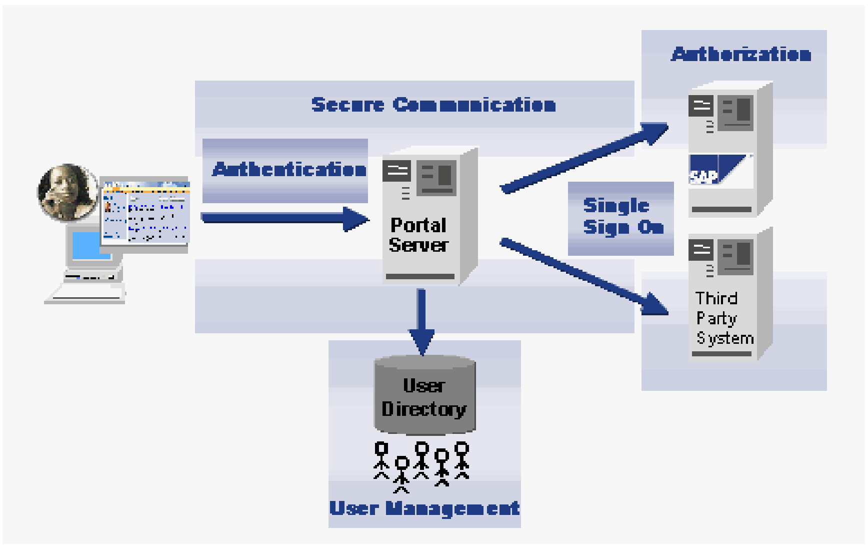


Figure 2.7 Portal security schemes

Authentication assures that the user which accesses the specific data source is really the one he claims to be. Authentication method refers to the type of logon information that must be forwarded to the server when the user tries to access some information resource. This process is usually preceded by identification level. This two level system has sense, since the closeness is achieved by *user identification*, while by user authentication (*user name/password*) portal gains users' confidence.

- *User Identification* - Portal should recognize user during the active session, even before he has actually introduced himself (e.g. entered his user name and password). A benefit that identification brings is fast access to the portal. Sometimes, various people can access personal information on a specific user. Some applications can therefore decide whether to use authentication or not. It is most probably desirable when a user is changing his/her preferences.

- *User Authentication* – User is requested to enter his/her credentials into system or show Kerberos ticket. This authentication is then applied to the rest of the session and relies on the existing organizational infrastructure.

Single Sign-On (SSO) is one of the key features of the portal that makes the user - system interaction easier. Once, when the user is authenticated by the portal, he/she can use the portal to access external applications. The user can actually access different systems and applications without need for any further authentication.

Authorization is the process of giving individuals access rights to a system based on their identity. Authorization level refers to who is allowed to set the user mapping to a specific portal data source. It is the actual check of the mapping permissions that were defined for the data source that the user is trying to access.

2.5.10.2 DIRECTORY

Through the portal, user can access various systems, applications and databases. Usually every of these systems has its own repository with information on users. The consequence of storing user data across different storages is redundancy, and therefore errors. For user data integration with the goal of user management many organizations utilize directories, based on *Lightweight Directory Access Protocol-u (LDAP),* as a central repository of user profiles. Directory is a specialized database adjusted for fast reading and search. It stores descriptive information, usually as attributes and supports advanced filtering techniques. Commonly, directories do not support complex transactions and *roll-back* schemes, but the replication ability with the goal of increasing availability and reliability, and decreasing the response times. *LDAP* is a lightweight protocol for directory service access that utilizes TCP/IP and other *connection-oriented* transfer services. *LDAP* information model is based on *entries*. One *entry* is collection of attributes that has a unique global identifier (*distinguished name* - DN). Every attribute of the entry has a type, and one or more values [LDAP02].

An active directory strategy is one of the key support elements for the portal, since most advantages they offer depend on the personalization of services for specific user. Existing directory of the organization is, frequently, primary storage of all the user information, required to tailor messages, information sources and authorization configuration. While integrating profile maintenance methods portal should use the existing directory functionality.

2.5.10.3 TRUSTED CONTENT

Trust is a critical factor for the growth of Internet content in the business environment. Information suppliers should, in order to gain user confidence, with the contents provide the certificate that the content is what it says it is while providing the information of where the content has come from and where it has been.

2.6 Elements of knowledge management

Portals deliver their users the promise of knowledge management providing them with the single point of access to the *people, places and things* that exist within and across their organization boundaries [MOR99].

People: People are among the most valuable assets of the organization. People retain critical business knowledge i.e. knowledge of processes, contacts and resources – that can make the difference between success and a failure of the project. On-line "people awareness" and *real-time* communication tools, provide active user means to view the status and on-line location of other users. Moreover, those tools provide means to engage users in instant on-line communication through *easy-to-learn* and *easy-to-use* interface.

Places: Places provide a context for work activities. Similar to the physical office space, an on-line workspace needs to be well equipped and well organized so that the people can focus on their task and not the tools needed to achieve it. Portals provide their on-line workspaces with *out-of-the-box* templates that can be used to immediately design and implement workspaces. Besides that, application authors can create templates, which meet their business needs and can customize details such as their look and access rights. When the places are instituted, they make it easy to non-technical users to perform their tasks and communicate with others. The result of their efforts — business plan or an RFP—then becomes a *reusable knowledge object* that can be accessed by others who may want to do similar work. Therefore, places not only provide the context for the knowledge development, but also a means for its preservation and ultimate reuse.

Things: Things are the essential resources or contents that people can access and utilize to perform business tasks. The reports, the documents, the web sites are all things that are used in daily business activities. This various types of content can be displayed and arranged in a meaningful context. In order to improve effectiveness of their personal and group-based efforts users can arrange things in their personal and community places.

People, places and things are fundamental components of a portal solution. By connecting people through *real-time* technologies, portals should improve the speed and accuracy of business decisions. Through collaboration features, communities can assemble, collaborate and innovate across geographical boundaries.

2.7 Benefits of the portal

Based on the preceding analysis the following benefits that portal brings can be identified:

- Users receive information through a standardized web-based interface.
- Users can mutually interact, query and share information right from their desktops.
- Portals integrate disparate applications and other data external to these applications into a single system. This system can share, manage, and maintain information from one user interface.
- Portals provide users access to external and internal data and information sources.

3 Personalization

3.1 Personalized system

Personalized web application is a hypermedia system, which adapts its content, structure and/or presentation of the networked hypermedia objects to each individual user's characteristics, usage behavior and/or usage environment. The term *hypermedia system* includes interactive system that enables users to navigate through the network of linked hypermedia objects, while latter contain a set of related content bearing elements of different media types, like text, images, video - clips, audio - clips, small applications and interaction elements (e.g. menus, buttons, checkboxes); they actually enable navigation through web pages [KOE01].

In contemporary web applications personalization commonly includes some of the following processes:

- *1-to-1 relationship* and other processes that *automatically* treat users differently depending on their demographic and static profiles (data collected in the registration process) or earlier interaction with the system. These processes are supported by the systems for *manual rule definition*, in which administrators define rules that determine what content is presented to which user.
- *Customization*, where users build their own user interfaces by selecting from different information channels. Actually, the user himself defines what he wants to see.
- *Recommendation* of new contents became irreplaceable part of personalized systems, as a tool for selection of the information relevant to the active user from the sea of information available on web. There are two generic approaches to the recommendation process: *clique-based filtering* and *content-based filtering*, as well as a number of hybrid systems that uses the advantages of both generic approaches with the goal to overcome their weaknesses.

All of the indicated personalization processes can be divided in three major tasks that are usually performed by different system components [KOE01]:

- *Acquisition* is the task of identification of all the available information about users' characteristics and computer usage behavior as well as about usage environment either by monitoring the computer usage or by obtaining the information from external sources. The collected data is afterwards delivered to the application adaptation component that will construct initial *user*, *usage* and *environment model*.

- *Representation and secondary inference* are the tasks to express the content of the user and usage models in a formal system, to allow access and further processing and to draw secondary assumptions about users and/or user groups, their behavior, and their environment, thereby integrating information from various sources.
- *Production* is the task to generate adaptation of the content, presentation and modality, and structure, based on a given user, usage and environment model.

3.1.1 Information Architecture of a Personalized System

Generally speaking, personalized system is a software product that applies rules to profiles of users and content to provide a variable set of user interfaces [INST00]. The information architecture components for a personalized system come from three areas of working context (determines the way the personalization happens); content and users (figure 3.1).

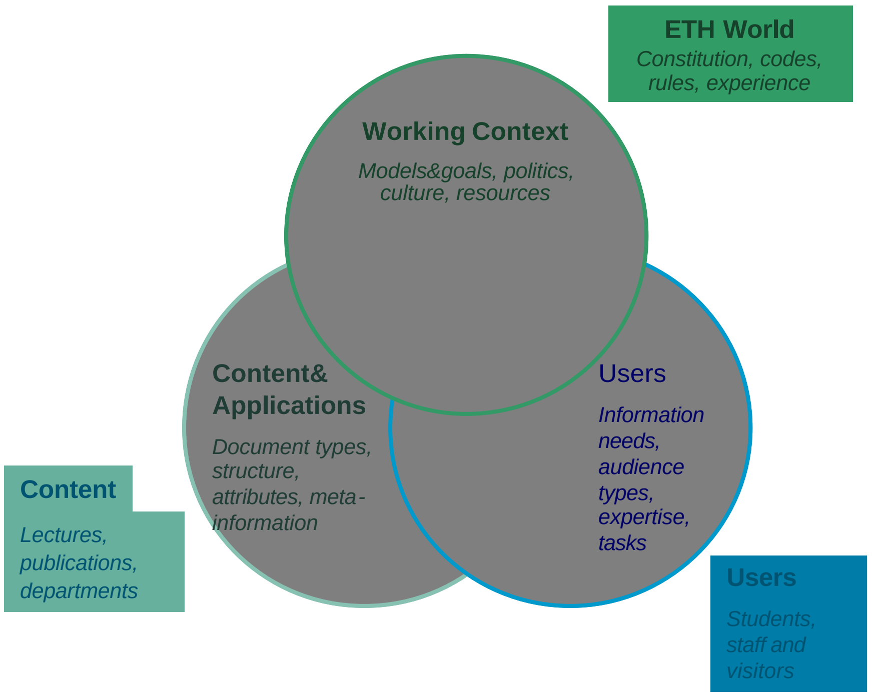

ETH World
Constitution, codes, rules, experience
Working Context
Models&goals, politics, culture, resources
Content& Applications
Document types, structure, attributes, meta-information
Users
Information needs, audience types, expertise, tasks
Content
Lectures, publications, departments
Users
Students, staff and visitors

Figure 3.1 Information architecture for personalized system

Personalization is a *data-intensive* task. One part of data can be observed by the system directly, while most others may require one or more additional acquisition steps.

3.1.1.1 USER DATA

Users have profiles that represent their interests and behaviors. Specific values for a profile are determined by the set of defined attributes and the possible values for each attribute. These attributes may include demographic data on user (as objective facts), assumptions on user's knowledge about concepts, user's skills and capabilities, interests and preferences, goals and plans.

In most of the cases, today's personalized web sites operate on the basis of the *demographic data* and purchase data only. The value of these data can be high when combined with high-quality statistic data. Among the rest it can contain record data (e.g. name, address,), geographic (e.g. area code, city, state…), user characteristics (age, sex …), registration for information offerings …

Assumptions on users' knowledge (or beliefs) about concepts, relationships between concepts, facts and rules with regard to the domain of the application system, are the ones among the most important personalization sources. Adaptation to a *user's knowledge* is a typical feature of intelligent tutoring systems.

Interests among users of the same application often vary considerably: information and advertisements targeted to one particular interest group may be of no interest to another. User interests are basic notion for so-called recommender systems (ref section 3.1.4.2).

User profiles are can be changed by *explicit* user actions (e.g. filling in the forms that request certain information about user profile) or *implicit* user actions, or example purchase of a product.

3.1.1.2 USAGE DATA

Usage data can be directly observed and recorded, or acquired by analyzing the observable data. The amount of data that can be collected this way in the system depends on the technical solution of that system. If the system is based on HTML then it is only possible to record what pages and files have been requested from the server, while in the case of Java applets, it is possible to record usage data on the level of mouse clicks and movements. There are several types of interaction that can be relevant for personalized systems. Typically, these are: selection actions, viewing times, rating, purchase and purchase connected activities on *e-commerce* sites (ref section 3.1.1.5 on the price of personalization).

Typically, a personalized adaptation is not direct product of the collected data. First the data is processed in order to discover initial contents of a user or a usage model. Usually, different kinds of patterns are acquired in the collected data, for example: frequency of repeating events, situation-action correlation, action sequences. Nevertheless, there are systems in which this information is utilized as user profiles.

The prerequisite step for any personalization technique (especially, recommendation) that mines usage data is identification of a set of server sessions from the raw data, which exists in web server log files. Ideally, user session gives an exact accounting of who accessed the web site, what pages were requested and in what order and how long each page was viewed. The two major difficulties in usage preprocessing are due to [MOB00]:

- *Local caching*; In order to improve performance and minimize network traffic most of the web browsers cache requested pages. The result is that when a user pushes the "back" button, web browser shows cached page and the web server is not aware of the fact that the user accessed it again.
- *Proxy servers;* Proxy servers provide intermediate caching level and create more problems. In a web server log file, all requests from the same proxy server have the same identifier, even when they come from different users. One of the consequences of the proxy level caching is that during long time period a lot of users can view a page that is a response to the single request.

The most reliable methods for user session identification from web server log files are *cookies* and dynamic URIs with embedded session IDs. However these techniques are not always available due to privacy problems and web server limitations. Independent of the type of usage mining, in order to be successful, systems that use them have to be publicly accepted. This is particularly important for *clique-based* filtering systems.

3.1.1.3 CONTENT

Content is profiled, too. It is described by the set of attributes and their assigned specific values. Content has features that can be used in the personalization process. These features can be: *metadata* (e.g. price, content author, producer, location where some service is available and so on), but they can, also, be features of the content itself (e.g. frequency of specific words in text, percentage of blue in the image or rhythm of some audio composition).

One part of the metadata on content can be changed explicitly by users (for example reviewing a movie with a "thumbs up" rating) or implicitly (for example, by tracking the purchase of a specific product and enough purchases could change the value of the attribute `product_popularity` from *average* to *hot* thereby affecting other users' experiences).

In the case of textual documents sets of attributes and their possible values are governed by a *controlled vocabulary*. For each attribute, there has to exist a consistent set of values used throughout the entire system. Sometimes creating the controlled vocabulary means deciding on the preferred term and changing anything indexed with a variation. This imposes the creation of a simple list of synonyms to link the different terms together.

3.1.1.4 PERSONALIZATION RULES AND ENVIRONMENT DATA

The most important attributes are those that apply to both user and content profiles. For example, if the site is intended for dog breeders, it is necessary to know which breeds each user owns. There exist contents and products specific for each breed, so this attribute will also need to be part of the content profile. Common vocabulary on dog breeds for both user and content profiles (i.e. controlled vocabulary for any domain) enables effective creation of personalization rules.

Personalization rules are based on the working environment, but as much on the quality of the resulting user experience. In the context of personalization, attributes and their values represent *glue* that connects users with the content and forms personalized user interface. Attributes of the content are compared to the attributes of the user. Specific attribute values about the user are matched with content meta-information to determine which content to display and how to present it at any given time.

3.1.1.5 THE PRICE OF PERSONALIZATION

Recommender systems became irreplaceable resources for users that look for intelligent ways for searching through large quantities of available information. The basic requirement for user interfaces of these systems is to *intelligently* guess user intentions or his interests and suggest contents consistent to those interests. Ideally, this system should determine user needs by simple interpreting sequences of user actions. The goal of most of the publicly available personalized systems is much more realistic: to understand simple user actions, including the mouse movements during the web page investigation or bookmarking the web page. Contemporary recommender systems are usually based on content ratings that are given *explicitly* by the users. Along with the explicit the *implicit* interest indicators are used.

When designing a personalized system technical solutions depend on various factors, including: the level of user engagement and adequacy of the reward, privacy, convenience for the user and the consequences of false adaptation, and the comparison between these loses and the benefits that recommendation brings.

I Explicit interest indicators

Explicit ratings are easy understandable, pretty precise and usual in everyday life. Although ratings exist in the free text form (e.g. book critics) the most common and obvious way of rating is based on discrete scale (stars for restaurants, rating from 1 to 10 in movie reviews). Ratings collected this way are used for further statistical processing in order to obtain average, range, or distribution of the values, etc. The gross of already implemented systems use this, explicit rating approach. However, this approach has several drawbacks [CLAY01]:

- User has to stop to enter explicit ratings therefore altering his normal patterns of browsing and reading.
- Ratings are subjective and dependent on users current information needs. However, if given in free text form (*annotations*) [GOLD92], they provide with the great amount of valuable information that can improve the quality of the recommendation.
- In most of the cases, though, entering the comments presents the additional effort for the user, which does not bring any benefit. This does not mean that the systems that provide the adequate reward do not exist. Nevertheless, the user will not rate the content if he does not see any direct benefit from the given effort. The question "who works, and who gets reward?" [GRU87] gains on significance. Logic consequence is that the unrewarded users continue to use the system, but stop rating the contents read.
- In most of the currently exploited systems the number of rated contents is much smaller than the number of examined.
- Users give negative ratings very rarely.
- The false recommendations are possible: content creators can produce piles of good recommendations for their own materials and bad for their competitors. Therefore, it is necessary that the recommender systems introduce precautions discourage the "vote early and often" phenomenon.

From the previously indicated facts it is obvious that the explicit ratings, though widely used, do not present so confident resource for recommendation as it is usually supposed.

II Implicit interest indicators

Implicit rating is a rating that is obtained by a method other than obtaining it directly from the user. Obvious advantages of this approach are:

- They can be gathered for "free".
- Every interaction between user and the system (even the lack of it) can be treated as a potential implicit interest indicator.
- Many different types of implicit interest indicators can be combined for more accurate rating.
- Implicit interest indicators can be combined with explicit ratings for an enhanced rating (countering, for example, "what I say is not what I want" problem).

Three broad categories of implicit interest indicators can be identified: examination, retention and reference [OARD98].

Table 3.1 Categories of an observed behavior [OARD98]

Category	*Observed behavior*
Examination	Selection, duration, editwear, repetition, item purchase or subscription
Retention	Save a reference or an object (with or without comments), (organized or not), print, delete
Reference	Object->Object (forward, reply) Portion -> Object (citation, hypertext link) Object -> Portion (cut&paste, quotation)

Examination : Selection actions (e.g. following a link) can indicate that the user is interested in a topic that can be reached by following that link; but it can also mean that the user needs an explanation of unfamiliar terms used on the current page (link to explanation) and so on. The disadvantage of this approach is lack of negative feedback, which consists in the fact that if the user does not follow one link when there exist other it does not mean that the user is not interested in that topic or that he does not need an explanation. The examination duration is potential interest indicator, too. However, measuring the effective time of examination is very difficult, since it is not possible to determine whether the user was in front of the monitor and examined the selected page during that time. In most of the cases examination duration is very bad interest indicator, but it can be used as a proof that the user is not interested in certain topic. If the presentation duration (and therefore maximum examination time) of a specific web page is shorter than some threshold then it is most probable that that page is not interesting to that user. *Editwear* and HEDO (history-enriched digital objects) [HILL94]) techniques sequentially store statistical data on menu selections, number of spreadsheet cell recalculations and time spent reading documents, in line-by-line manner summing over sections and whole documents. Repetition is another behavior from the examination category and refers to the repetition of any interaction between user and system; similar to the conventional library usage, where timestamps with the date of borrowing/returning the book are used. Purchase and purchase connected activities on e-commerce web sites are strong interest indicator; however, there is no 1-to-1 mapping between items purchased and interests of the user, since customers can buy gifts for other people. Other activities like, for example, putting the products into virtual shopping cart (for later purchase), participating quizzes, product registration, … do not show user interests as much as the act of buying itself.

Retention: This category groups all those behaviors that suggest some degree of intention to make further use of an object. *Bookmarks* are one of the most common ways of indicating an interest. Unlike mouse clicks, that can also be unintentional and hardly require any effort or investment, bookmarks are the result of an intentional act, something that (especially if the bookmark is classified and put into separate folder) represents an entry for inference process [RUCK97]. Printing belongs to this category due to the permanence of the printed page, though users can print documents and images to facilitate the examination process or to forward them to other individual. Finally, this category is distinguished by the possibility of directly observing evidence of negative evaluations. When the retention is a default condition, as in the most of the e-mail systems, a decision of the user to delete an object might support to an inference that the deleted object is less valued than other objects that are retained.

Reference: This category groups all those activities that have the effect of establishing some form of link between two objects. Forwarding a message, for example, establishes a link between the new message and the original. Similarly, traditional academic citations, as well as less formal hyperlinks in a web page or complicated links between *Usenet* news articles, belong to the same category.

3.1.2 Rule-based personalization

Architecture of the framework supporting rule-based personalization is illustrated by figure 3.2. In this framework users and content meet at the *user interface* layer through the process of personalization.

Underneath the user interface is the *profile layer*. On this layer the specific values for the attributes are used to determine what content to present to which user under what conditions. Here exist: user profile and content profile that are matched among them through the set of rules.

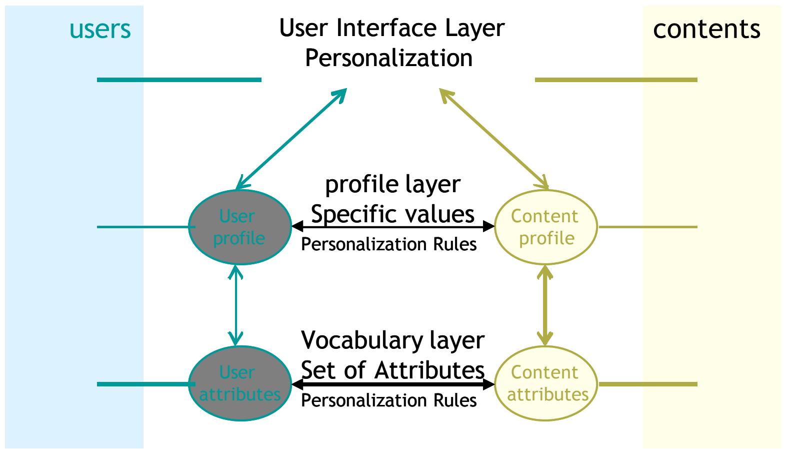


Figure 3.2Rule based personalization

The basic layer is the *vocabulary layer*, on which the assignment of attribute values is regulated. On the vocabulary layer, attributes are defined and the set of acceptable values (preferred terms) are specified. The relationships between attributes are defined. Both, users and contents have their own attributes, but they are coordinated to make sure that the higher-level profile information is synchronized. The vocabulary is actually the set of attributes and values, while a profile is merely one specific instance of the vocabulary.

Personalization rules leverage the profiles, attributes and values in order to make the personalized user experience. The most effective rules operate on the set of attributes as a whole, at the vocabulary layer. When user and content profiles share the same attributes, then it is possible to make rules work for all values of those attributes. The example of one such rule could be: `show all CD's by this user's favorite artist`. If the user profile has an attribute `favorite artist` that shares the same values as the content profile's attribute `sung by`, then it is possible to form general-purpose rule that works for all values. If it is not possible to do the profile-layer rules, then it may be possible to make a series of rules based on each value: `If the favorite_artist is` *Elvis*, `show CD's sung_by` *Elvis Presley*. This would, however, become very inefficient. Since the number of personalized systems and portals support this type of personalization, the rules are usually categorized into:

- *Classification rules*, which determine the conditions under which the contents are changed. They refer to the current resources (for example, the ones stored during the active HTTP session). One classifier can define multiple classes. For example, `CustomerLevel` classifier defines that the user belongs to the gold order in the airline loyality scheme (there are blue and silver, more) if he/she collects more than 100 points.

```
CustomerLevel classifier:
  CustomerLevel is Gold when
  currentUserProfile.enterprise is greater then 100...
```

- *Action rules* enable querying the resources that provide contents fulfilling the given conditions as a result. For already illustrated airline company site, example action rule `Exclusive` presents exclusive offers (items containing keywords like `tropical`,...)

```
Exclusive action:
  Select content whose Article.keywords includes value tropical,...
```

- *Connection rules* are used to connect actions and classifiers. Depending on user classification this rule calls on specific action to show specific contents. Therefore, for example, low budget action can be shown to the blue order member, and `Exclusive` to a `Gold` order one. Common name for these rules is content targeters.

```
When CustomerLevel is Gold do Articles
```

3.1.3 Customization

Attribute values can be set *manually* (user or system administrator), and *automatically,* using some software process. The way personalized system supports setting profiles is very important, since being completely manual; it requires too much work in maintenance process.

Customization is the simplest personalization approach, where users define personalization rules; actually by explicit actions they change system contents and layout. It is also called *checkbox* personalization [MUL00]. Eventhough a user has the freedom to adapt an application to him, by choosing some of the available information sources, at the same time; he bears the entire adaptation load since he has to state his interests clear and precise. This includes the fact that the user understands and knows information (or at least its structure) that can be found on the site, in order to be able to choose the right information from the set of offered alternatives. If One user adapts the system to his personal needs then there is no way for him to share these adjustments with another user, without having corresponding administrative rights. Another user would have to go through exactly the same procedure in order to get the same adjustments.

A number of sites receive information about user needs and their preferences through HTML forms. Forms collect information that are rarely changed (e.g. name and address of the user) and information that should be updated from time to time (for example, the background color and selected information channels).

3.1.4 Recommendation

In everyday life it is often necessary to make choices without sufficient personal experience of the alternatives. Always when they are not sure about their choices people rely on recommendations from other people, either by word of mouth, recommendation letters, movie and book reviews printed in newspapers or general surveys. Recommender systems assist and augment this natural social process [RES97]. Recommender systems exploit ratings produced by the entire user population in order to reshape an information space for the benefit of one or more individuals [OARD97]. In a typical recommender system its users produce recommendations (they state their preferences and opinions), which are collected by the system and transferred to the users that need them. In some systems primary transformation is the aggregation process, while the value of other systems lays in the ability to connect users that recommend to the ones that need these recommendations [RES97]. In the last years, interest into recommender systems grew especially under the influence of personalized Internet applications. Currently a large number of *e-commerce* web sites use recommender systems in order to personalize their contents and marketing campaigns. Even the biggest on-line retailer *Amazon.com* uses the recommender technology.

Problem of recommending items from some fixed database has been studied extensively, and two main paradigms have emerged [BAL97]:

- *Content-based filtering* is an approach to recommendation which is used to recommend items similar to those that a given user has liked in the past.
- *Clique-based filtering* is a two step approach: first the users whose tastes are similar to those of given user are identified, and then this user is recommended the items they have liked.

In the following sections, in more detail, both generic approaches are described; together with hybrid systems, which arose with the goal to use advantages of those approaches, but at the same time for solving their disadvantages.

3.1.4.1 CONTENT-BASED FILTERING

Filtering interesting contents in order to produce recommendations has its roots in the *information retrieval* (IR), and therefore uses a large number of techniques from this area. The assumption that underlies this approach (in further text it will be referred to as IR approach) is that the content of a page, not its layout, interac tion ability, or download speed, present user interests. In this manner, for example, text documents are recommended based on a comparison between their content and a user profile. Data structures used for representation of both information sources are created by extracting features from their text (*vector-space model*, refer to section 3.2.1). Very often some weighting scheme is used, which gives high weights to discriminating words. After the pages that match user profile have been picked, they can be presented to the user and feedback of some kind elicited. If the user found some page relevant, weights for the words extracted from it can be added to the weights for the corresponding words in the user profile. By being simple and fast, this approach improves the quality of results in IR applications. There are many alternative methods for weighting words or other features from the text and for updating user profiles. This kind of system has several disadvantages:

- Primarily, only the shallow analysis of certain types of contents is possible. Existing methods for feature extraction are not applicable on contents from some domains (such as movies, music…). Even textual documents representations capture only certain aspects of the content, though there are many others that could influence a user's experience. In the case of web pages, for example, IR methods completely ignore æsthetic qualities, all multimedia objects (even text embedded in figures) and network factors, like loading time, for example.
- It is not possible to compare contents of different types, since comparison is based on features, which are more physical than essential. For example, it is not possible to compare the RGB specter density of some image and frequency of appearing some word in text or average sound intensity of some music sequence, even when they describe same event.
- Additionally, it is not possible to filter out contents based on their quality, style or point of view. For example, system is not able to differentiate well from badly written article even when they refer to the same topic.
- Problem of over-specialization is present in this area, too. If the system can recommend only items that are very similar to the user profile, then the user is restricted to seeing items similar to those already rated. This problem is often addressed by injecting a note of randomness, for example, the crossover and mutation operations.
- The user feedback is elicited, in order to model his interests.

3.1.4.2 CLIQUE-BASED FILTERING

Unlike IR approach that recommends contents based on their similarity with user profile, this approach recommends to an active user all those contents that similar users found relevant. Actually, instead of looking for contents similar to user interests, users with similar interests are looked for. Usually, for each user a set of similar neighbors is identified. These are the users whose previous content ratings are strongly correlated to the ratings of the active user. Then, based on those ratings it is predicted how much the new items will be interesting to that active user. The only thing that is required to know about contents is their unique identifiers. In general, this process consists of the following three steps [KOE01]:

- Find similar neighbors; standard similarity measures (*vector-space model*) are used to compute the distance between the active user's representation (such as a feature vector) and representation of the set of users.
- Select a comparison group of neighbors; when the distances to other users are available the *set of closest users*, the selection can be made in several ways. If the selection is made based on a fixed threshold, then all users within given distance are taken with the possible risk of obtaining too few or too many closest users. The other type is selection of fixed number n of closest users, where there is a risk that some of these may be poor matches or that users are ignored that could add the quality of prediction.
- Compute prediction based on (weighted) representations of selected neighbors. Different approaches are possible to determine whether a particular neighbor is likely to be a good predictor. Most of these evaluations are based on statistical measures, such as overall rating average of the person, the person's average deviation from the group mean and the correlation of deviations between the target user and the neighbor.

This process is also known as collaborative [GOLD92] or social filtering [SCHAR95] (in the text to come it will be referred to as approach). The term *collaborative filtering* was used for the first time in the description of the ***Tapestry*** system and in that context it determines the process in which "people cooperate to help one another perform filtering by recording their reactions to documents they read". The term in use, however, is not completely appropriate having in mind that this approach has only few collaborative characteristics. This type of filtering solves the limitations of the IR approach since the consequence of using only the ratings of other users gives the possibility to work with different types of contents and recommends user only those contents that are completely different from the contents the user have already seen; the performance of the system does not deteriorate with the growth of the user population, since there exist *number of users* times more ratings. However, this approach introduces certain problems of its own:

- *Problem of new content*: if a new item is appears in the database there is no way it can be recommended to a user until more information about it is obtained through another user either rating it or specifying which other contents it is similar to.

- *Scarcity*: since users have limited resources to experience contents (read articles, see movies, and listen to music) the density of user ratings on items decreases. Pit becomes less likely that the significant number of a user's neighbors will have experienced the item for which a prediction is being requested [OCONN99].
- *Problem of a new user*: the system does not contain any data on a user that did not rate any of the contents. This means that the system is not able to recommend new content to that user, since it does not recognize him i.e. does not know his interests. The problem of a new user is extensively studied and number of techniques for solving it already exists. Some of these techniques are described in [RAS02].
- *Problem of unusual interests*: a user whose tastes are unusual cannot be compared to the rest of the population, since there will not be any other user who is particularly similar; this leads to poor recommendations.

It is obvious that these two approaches are complementary i.e. that the second solves problems the first one brings and vise versa. Therefore a significant number of systems combine them in order to generate better and more useful recommendations. As already indicated the similarity of contents and users, as well among users themselves can be determined in a number of ways, by using IR techniques, statistic, machine-learning, data mining and artificial intelligence methods and so on. In the following sections some of the most frequently used methods will be described, while a detailed review can be found in [KOE01]. Having in mind that the problem of this master thesis is the university itself a variety of contents is practically unlimited. Still, there exist similarities in user behavior. Therefore, the focus will be on CF techniques and systems that use them.

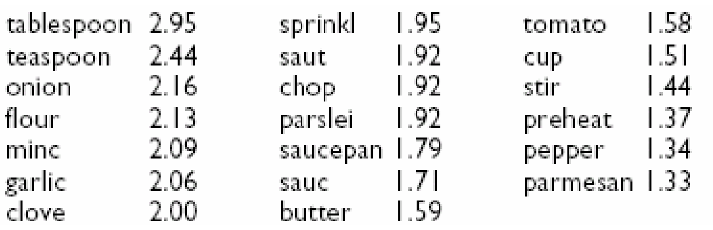
3.2 Methods for detecting similarity

3.2.1 Vector-space model

Vector-space model from the area of information retrieval [SAL83] provides an appropriate representation for documents based on their constituent words. This model has been extensively studied and frequently used; it presents a basis of all commercial web search engines and has shown to be competitive with alternative IR methods. The assumption is that the content of the page can be presented by only using the words contained in the text, while ignoring all mark-up tags, images and other of multimedia information. In this model both documents and queries are represented as vectors, where: w is a web page representation, m is a representation of user's interests, $r(w,m)$ is a function to determine the pertinence of a web page given a user's interests, and $u(w, m, s)$ represents a function returning an updated user profile m based on feedback information s on page w.

3.2.1.1 TFIDF

Assume some dictionary vector *d*, where each element d_i is a word. Each document then can be presented by a vector *w*, where element w_i is the weight of the word d_i for that document. The figure 3.3, for example, shows a profile of a document that contains text on cooking.



tablespoon	2.95	sprinkl	1.95	tomato	1.58
teaspoon	2.44	saut	1.92	cup	1.51
onion	2.16	chop	1.92	stir	1.44
flour	2.13	parslei	1.92	preheat	1.37
minc	2.09	saucepan	1.79	pepper	1.34
garlic	2.06	sauc	1.71	parmesan	1.33
clove	2.00	butter	1.59		

Figure 3.3 Example of a content based profile

If the document does not contain d_i then $w_i = 0$. The words are then reduced to their *stems* using the Porter algorithm [PORT80], words from standard stop list are ignored (in English dictionary there are 571) and TFIDF weights are calculated: weight w_i of a word d_i in the document *W* is given by:

$$w_i = (0.5 + 0.5\frac{tf(i)}{tf_{\max}}) \cdot \log\frac{n}{df(i)},$$

where *tf(i)* is the number of times word d_i appears in document *W* (*the term frequency*), *df(i)* is the number of documents in the collection that contain word d_i (*the document frequency*), *n* is the number of documents in the collection, and tf_{max} is the maximum term frequency over all words in *W*. The experiments, described in [PAZZ96], have shown that too many words lead to poor performance when classifying web pages using supervised learning techniques; therefore, usually, 30 to 100 words are used. When, for example, first 100 words are picked *w* is normalized to be of unit length, to allow comparison between documents of different lengths.

3.2.2 Cosine distance

As already indicated, vector representation is used for modeling web pages; it is also utilized for modeling user interests, user profile *m* (that corresponds to a query in retrospective IR system). In order to measure how well the document *w* matches a given profile, the variant of standard IR cosine distance $r(w,m) = w \cdot m$. is used. Updating *m* also corresponds to a normal operation in the retrospective IR system: relevance feedback [ROC71]. A simple updating method can be used: $u(w, m, s) = m + s \cdot w$.

3.2.3 Euclidean and Manhattan distance

If two vertices in n-dimensional space correspond to, for example, two users represented with sets of their ratings $R_a=(r_{a,1},r_{a,2},\dots,r_{a,n})$ and $R_u=(r_{u,1},r_{u,2},\dots,r_{u,n})$ and values for the weights for each dimension $W=(w_{a,1},w_{a,2},\ldots,w_{a,n})$ determined then Euclidean distance and weighted Euclidean distance can be calculated using the following equations:

$$d(R_a,R_u)=\sqrt{\sum_{i=1}^{n}\left(r_{a,i}-r_{u,i}\right)^2} \qquad i \qquad d(R_a,R_u)=\sqrt{\sum_{i=1}^{n}w_{a,i}\cdot\left(r_{a,i}-r_{u,i}\right)^2}$$

Very often Manhattan distance is used: $d(R_a,R_u)=\sum_{i=1}^{n}\left(\left|r_{a,i}-r_{u,i}\right|\right)$.

3.2.4 Pearson correlation

Pearson correlation is used for measuring the level of existence of linear dependency between two variables. Pearson correlation coefficient is derived from linear regression model where following assumptions are valid: dependency is linear, the errors are mutually independent, mean error value is equal to 0 and variance is constant for every combination of independent variables. If all those requirements are not fulfilled then Pearson correlation becomes inadequate similarity indicator. ***GroupLens*** [KONS97] is the first system that used Pearson correlation to weight user similarity and computed a final personalized prediction for Usenet news articles. A final prediction is calculated as a weighted average of deviations from the neighbor's mean:

$$p_{a,i}=\overline{r_a}+\frac{\sum_{u=1}^{n}(r_{u.i}-\overline{r_u})\cdot w_{a,u}}{\sum_{u=1}^{n}w_{a,u}},$$

where $\overline{r_a}$ corresponds to the average rating of the active user, $p_{a,i}$ represents the prediction for the active user a for item i, n is the number of neighbors and $w_{a,u}$ is the similarity weight between the active user and neighbor u is defined by Pearson correlation coefficient:

$$w_{a,u}=\frac{\sum_{i=1}^{m}(r_{a.i}-\overline{r_a})\cdot(r_{u.i}-\overline{r_u})}{s_a\cdot s_u}.$$

3.3 Methods for Knowledge Discovery

3.3.1 Data Mining

Data Mining is a process of extracting information or knowledge from the data set with the goal of decision making. Last few years this area has become very popular as a consequence of existence enormous amounts of data. Information and knowledge discovered during the *data mining* process can be used in different areas, from scientific to market analysis. Unlike information retrieval and information access techniques, which have the goal of helping users find documents or data, the goal of *data mining* is to discover useful knowledge by analyzing correlations that exist in data by using prosperous *data mining* techniques. Knowledge that can be collected during a *data mining* process includes: *concept description, association rules, classification and prediction* and *clustering*.

Very often *data mining* is referred to as knowledge discovery in databases (KDD), even though this technique is only a part of the knowledge discovery process that includes the following steps:

- *Data cleaning*: in this step *dirty* data is cleaned. The *dirty* data includes: *incomplete data* (attributes or attribute values missing), *noise* (attribute values that are not correct and/or not expected), *inconsistent data* (discrepancy between attributes and their values).
- *Data integration*, where data from different data sources is combined. Among those sources there are numerous databases, with different contents and data formats. Inconsistency between formats leads to redundancy and disagreements between attributes and data.
- *Data transformation*; data is transformed into corresponding and agreeable formats using methods like aggregation, normalization and equalization.
- *Reducing the amount of data*, the amount of data is reduced, but at the same time retains integrity of the original data set. Different strategies are used, for example: *data cube aggregation* (i.e. `sum()` and `min()`), *dimension reduction* (filtering out irrelevant data), *data compression* (substitution of data values with the encoded data), *precision reduction* (substitution of data values with alternative smaller representations) and *generalization* (substitution of data values on lower conceptual level by values on higher).
- *Data mining* – intelligent pattern recognition.
- *Pattern evaluation* – identification of interesting patterns that represent knowledge, using techniques that include statistical analysis and query languages.
- *Knowledge representation* – discovered knowledge is represented using techniques for data visualization and knowledge representation, for example charts, maps, tables and rules.

The first four steps are often referred to as preprocessing or data preparation.

3.3.1.1 Concept Description

Very often the condensed reviews of derived knowledge, which cannot be obtained by simple quotation or data manipulation, are needed. For this usually the description of concepts is utilized. This technique includes:

- Description analysis that as a result provides short review of data set, and
- Discrimination analysis that as a result gives a revised comparison of two data sets.

During the concept description process data, important for the execution of the given task, is generalized and summarized in order to transfer it from lower to higher conceptual level, and at the same time, in order to increase comprehension it is compressed by using smaller number of description terms. Data generalization is possible to do by using data cubes or *attribute-oriented induction (AOI)*. AOI approach includes attribute generalization or their removal.

3.3.1.2 Association Rule Discovery

This is the process of discovering interesting correlations in large datasets. Process of making marketing decisions or web site design can be considerably eased up if, for example, interesting relations in the large set of business transactions is identified. Typical example of market basket analysis that analyzes customer's buying habits by discovering associations between articles that he bought. The following example illustrates the rule discovered among transactions in computer book store and denotes that the customers that buy books on operating systems at the same time buy books on Linux operating system:

```
Operating System => Linux [support=4%, confidence=48%]
```

Support of 4% denotes that in the 4% of all transactions books on operating systems and Linux were purchased together.

Confidence of 48% determines that the 48% of customers that bought the book on operating system bought the book on Linux, too. Association rules are considered only if they satisfy two user defined requirements: minimum support threshold and minimum confidence threshold. The association rule discovery is a two step process:

- First step is the discovery of all *frequent itemsets* that fulfill min-δ, predefined support value;
- And then follows the discovery of association rules that fulfill the min-α, predefined value for confidence, from the frequent itemsets discovered in the first step.

This method is often referred to as one-dimensional association rule discovery. Beside this one, there are multidimensional, multi-level, quantitative (distance lead) and boundary-lead association rule discovery.

3.3.1.3 Classification and Prediction

These two techniques are used for data classification and trend prediction. In the classification process the class labels are determined while during the prediction process the continual functions are discovered. Classification is a two step process that includes:

- *Learning process*, during which the model is used to analyze learning set. This set consists of training tuples and items randomly selected from the population. Having in mind that the class labels are assigned to every possible tuple prior to the learning process it is often referred to as *supervised learning*.

- *Classification:* the learned model is used for classification. The accuracy of the classification process is estimated by using randomly selected labeled items (test set). Predicted labels are compared with the assigned and the accuracy is defined as a percentage of test examples that are correctly classified. If the percentage is satisfying then the model can be used for classification of new items. If it is not, the model is adapted until it reaches the acceptable prediction accuracy.

The most common data classification techniques are:

I k-Nearest Neighbor

Each item from the learning set is interpreted as a vertex in n-dimensional item space (for example, the ratings of the active user). This algorithm classifies unclassified items by detecting its k nearest neighbors, based on the chosen metrics. This is a three step process:

- The weights for all users are determined based on their similarity with the active user.
- For the specific item the subset of the user population is selected in order to be used in the prediction process.
- The ratings are normalized and the prediction value is calculated as a weighted average of dimensions (for example ratings) of the selected neighbors.

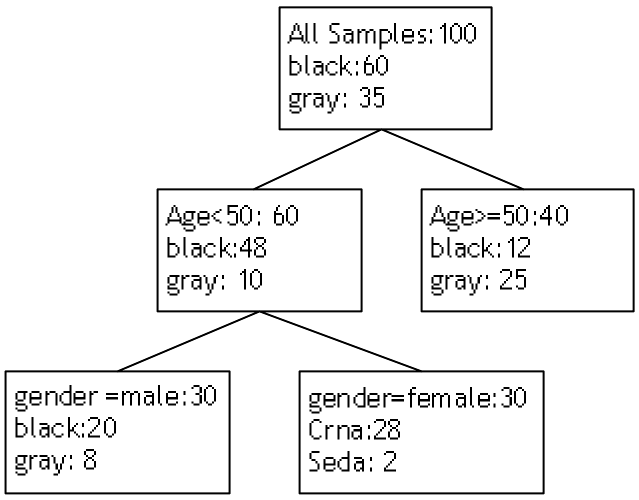
For nearest neighborhood determination ***Euclidean*** or ***Manhattan*** distance can be used. However, the most often ***Pearson*** correlation is used. Unclassified item is assigned to the most commonly used class in the k nearest neighbors.

This is a slow learning technique and it requires efficient indexing techniques.

II Decision Trees

These are the trees whose internal nodes represent conditions, and leaves correspond to the classes of data. The conditions in the internal nodes usually have a small number of possible outcomes. Every branch in the tree corresponds to the decision that can be made during the condition evaluation. If the branching through the tree is performed based on the results of every test, the leaf that contains the label for the class for the given item is reached. The decision trees can very easily be converted into classification rules by using *if-then* construction.

The figure 3.4 shows the simple example decision tree. Let assume that each item is described by the set of attributes: gender, age and hair color. After the training process column *gender* should contain three possible values: male, female and undefined; the same works for the *hair color* column: black, gray and undefined. The following decision tree is formed for hair color prediction.



Bayesian classifier predicts that unclassified sample $X=(x_1,x_2,\ldots,x_n)$, belongs to the one of m classes $C_1,C_2,\ldots,C_m$ having the highest posterior probability, conditioned on X i.e. if and only if $P(C_i|X)>P(C_j|X)$ for $1=j=m, j?i$. As $P(X)$ is constant for all classes only $P(X|C_i)\cdot P(C_i)$ need be maximized. Probability that the sample belongs to the class C_i is estimated to $P(C_i)=s_i/s$, where s_i is the number of training samples of class C_i, and s is the total number of training samples. Given data sets with many attributes it would be extremely computationally expensive to compute $P(X|C_i)$. Therefore the assumption on conditional class independence is made. It presumes that the values of attributes, for the given class, are independent from one another thus $P(X\mid C_i)=\prod_{k=1}^{n}P(x_k\mid C_i)$. The probabilities $P(x_1|C_i)$, $P(x_2|C_i)$, …, $P(x_n|C_i)$ can be estimated from the training samples, where:

- If the attribute A_k is categorical, then $P(x_k|C_i)=s_{ik}/s_i$, where s_{ik} is the number of training samples of class C_i having the value x_k for attribute A_k, and s_i is the number of training samples belonging to C_i.
- If A_k is continuous-valued, then the attribute is typically assumed to have a Gaussian distribution.

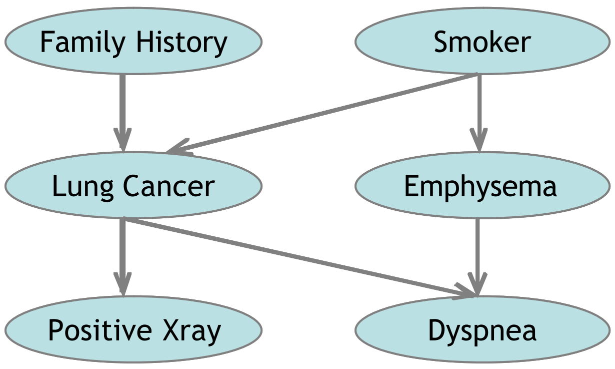
In order to classify an unknown sample X, $P(X|C_i)\cdot P(C_i)$ is evaluated for each class C_i. Sample X is then assigned to the class C_i if and only if $P(X|C_i)P(C_i)>P(X|C_j)P(C_j)$ for $1=j=m, j?i$. In theory, compared to other classifiers Bayesian classifiers have the minimum error rate. Various empirical studies have shown that this type of classifier can be compared to decision trees and neural networks in some domains.

Bayesian belief networks: These networks are used if there exist dependencies between variables and they specify joint conditional probability distributions. They provide a graphical model of causal relationships, on which learning can be performed. In this direct acyclic graph each node represents a random variable, while each arc represents a probabilistic dependence. If an arc is drawn from node Y to a node Z, node Y is a parent or immediate predecessor of Z. Each variable is conditionally independent of its non-descendents in the graph, given its parents. The variables may correspond to actual attributes given in the data or to *hidden variables* believed to form a relationship. The figure 3.5 shows a simple belief network with six Boolean variables. The arcs allow a representation of causal knowledge. For example, having lung cancer is influenced by person's family history of lung cancer, as well as whether or not the person is a smoker. Furthermore, the arcs show that the variable *LungCancer* is conditionally independent of *Emphysema*, given its parents. This means that once the values of *FamilyHistory* and *Smoker* are known, then the variable *Emphysema* does not provide any additional information regarding *LungCancer*. The second component defining a belief network consists of one *conditional probability table* (CPT) for each variable and specifies conditional distribution $P(Z|Parents(Z))$. Figure 3.5 b shows CPT for *LungCancer*, containing the values for the conditional probability for this variable given each possible combination of values of its parents. For example:

$$P(LungCancer='yes' \mid FamilyHistory='yes', Smoker='yes') = 0.8$$

The joint probability of any tuple $(z_1,\ldots,z_n)$ corresponding to the variables or attributes $Z_1,Z_2,\ldots,Z_n$, is computed by $P(z_1,\ldots,z_n)=\prod_{i=1}^{n} P(z_i \mid Parents(Z_i))$. A node within the network can be selected as an *output* node, representing a class label attribute. There may be more than one output node. The dassification process, rather than returning a single class label, can return a probability distribution for the class label attribute that is, predicting the probability of each class.

a)



b)

	FH,S	FH,¬S	¬FH,S	¬FH,¬S
LC	0.8	0.5	0.7	0.1
¬LC	0.2	0.5	0.3	0.9

Figure 3.5 Simple example of a Bayesian belief network [HAN00]

The network structure may be given in advance or inferred from data. The network variables may be *observable* or *hidden* (*missing values, incomplete data*) in all or some of the training samples. The network training process consists of computing the CPT entries. When the network structure is given and some of the variables are hidden, then a method of gradient descent can be used to train the belief network (a detailed can be found in [HAN00]).

Beside Bayesian belief networks there are neural networks that use *backpropagation*, generic algorithms, based on a natural selection paradigm as a learning method, fuzzy sets...

3.3.1.4 CLUSTERING (SEGMENTATION)

Cluster is a collection of mutually similar objects, while segmentation (*clustering*) is a process of grouping data into segments. Unlike the classification process, where class labels for every object are given in advance, in clustering these class labels are not previously known. Hence, this method is often referred to as unsupervised learning. Usually, when this *data mining* model is used it is not obvious neither what is looked for nor what can be found. The goal is to discover a distribution and correlations that exist among objects by identifying crowded and scarcely populated regions. Segmentation is used in the cases where large amounts of data are available, and that data contains highly organized logical structure and numerous attributes. The results of the clustering process enable:

- *Relationship visualization*: One of the great advantages of this method is the simplicity of generating the graph or map from the model. A visual display enables the user or an operator to catch the similarities that exist among data records, on the first glance.
- *Emphasizing anomalies:* Graphs, also, simplify noticing the records that do not fit into model.
- *Creating samples for other data mining action*: A number of *data mining* algorithms, for example decision trees, and during the analysis requires a number of cases. Off course, if the number of cases is very large decision trees become immense, therefore the clustering is used to separate cases that will later be used for decision tree generation.

Unlike decision trees, this method is difficult to interpret and requires a lot of experimenting, in order to provide clusters that bear some meaning. The basic disadvantages of this algorithm are that the final results of this process are difficult to understand, since there are no rules that describe them, similar to those that exist in decision trees, as well as the difficulties that appear while comparing data of different types. Due to this quality this algorithm is very rarely used for the discovery of information, which later will be used directly in the decision making process. More often it is used to generate groups of records that can later be studied using some other methods, like decision trees, for example. The most popular clustering techniques are:

I Partitioning

Using this method the set of n objects is partitioned into k groups that are referred to as partitions where each one of them represents one cluster. It is assumed that the number of partitions is smaller then the number of objects, that each partition contains at least one object and that each object has to belong to some partition. The process is iterative and starts by initial partition creation, followed by the cleaning process that utilizes iterative relocation techniques (IRT). IRT improves the quality of partitions by moving objects from one group to the other. Different heuristic methods are used in order to avoid detailed pass through all possible partitions, including:

- *k-means method:* each cluster is represented with the average value of all the cases that belong to that cluster. This algorithm works only with numeric values.
- *k-medoids method:* each cluster is represented by the case that lies nearest to the center of that cluster.
- *k-modes method:* extends *k-means* method so that it can cluster categorical data types, too.
- *k-prototypes method:* is a combination of *k-means* and *k-modes* method, which enables clustering of hybrid data types [HUA98].

K-means is one of the most frequently used clustering methods. Almost every commercial *data mining* application implements some of the variations of this algorithm. The basic qualities of this algorithm are [HUA98]: it has to be efficient while processing vast data sets, usually ends in the local optimum and formed clusters are convex. Basic disadvantage of *k-means* clustering algorithm is that the value of variable *k* has to be known in advance. There are lot of variants of this algorithm, which differ among themselves in a way of initial cluster determination and similarity measuring function.

II Hierarchical clustering

Hierarchical methods enable hierarchic decomposition of the given set of cases using:

- *Division,* the start assumption is that all cases lie in the same cluster, and then that cluster is divided into smaller parts until the condition of the end is reached. This is a *top-down* approach.
- *Accumulation,* the start assumption is that every object is one cluster, and then similar clusters are joined together until the condition of the end is reached (*bottom-up* approach).

III Clustering based on the distribution density

These methods discover clusters by expanding the start cluster until the neighborhood density overtakes a certain threshold. If the start cluster is *c*, the density threshold requires that each object's neighborhood (predefined radius) contains minimum number of cases. Therefore, these methods may discover clusters of various forms and may be used to measure the growth of the cluster when automatic or interactive analysis is used.

There are many other different clustering techniques, including *grid-based* methods, *model-based* methods, and many other hybrid methods.

3.3.1.5 Microsoft Clustering

This algorithm is based on the *Expectation-Maximization* (EM) algorithm. It is a two step method. In the first step (*E*) the cluster membership of each case is determined; while during the second step (*M*) the parameters of the models are reestimated using this cluster membership. *Microsoft Clustering* is similar to *k-means* method, which uses randomly selected records for cluster centers. In the clustering process records are represented as vertices in multidimensional space, in order to determine distances between them. The assumption is that the similar records are situated in the neighborhood. Similar to the countries on the map, clusters have boundaries that surround cases that reside in the same cluster. Model generated by the clustering algorithm, also contains pointers to coordinates that locate the case in the space. This coordinates point to other records that belong to the same segment.

- Randomly selected cases are used as start points. If k is assigned value 7, then seven points are randomly selected and are temporarily assigned cluster means.
- Assign cases to each mean using some distance measure.
- At the same time compute new means based on members of each cluster This is, usually, achieved by simple computation of a mean value for the cluster and taking the vertices that are the nearest to that mean value for new cluster means.
- This procedure is cycled until convergence.

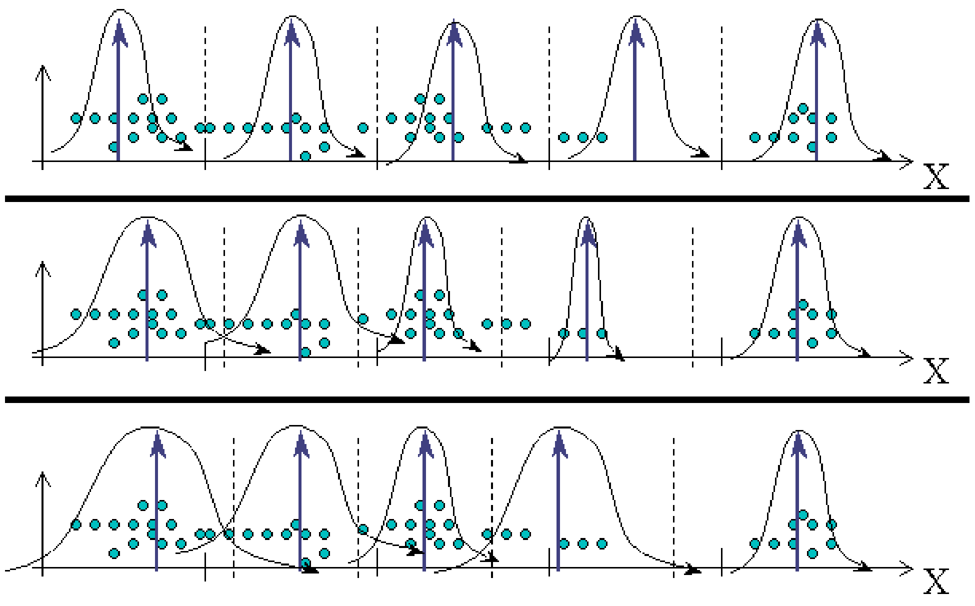


Figure 3.6 A simplified display of few iterations of MC algorithm

Since the initial values for the cluster means are randomly determined, it is very probable that when same data is reprocessed completely different clusters are generated. Due to the continual adjustment of cluster means, cluster boundaries are also shifted. The cluster boundaries connect points that stand halfway the cluster means distance. As a consequence of constant cluster boundaries shifting the cases that once belonged to one cluster can be found in the other. Figure 3.6 displays a few iterations of the EM algorithm for a one-dimensional dataset. The data in each cluster is assumed to have a Gaussian distribution. After each iteration, clusters' means are shifted. The basic distinction between EM and k-means algorithm is that EM has no strict boundary among clusters. A case is assigned to each cluster with a certain probability.

Previously described procedure is very simple and easy to understand when attribute values are easy measurable and mean values are possible to calculate and that way cluster membership determined i.e. numeric values. Usually, the situation is completely different. People find similarities between objects without using any mathematical calculations. Therefore, the method of translating these subjective comparisons into numbers has to be found in order to prove similarity. The data in databases is often stored in formats that are not easy to map into numbers, for example colors, geographic locations, automobile makes, languages, animal species. One possible solution is to connect each of the attribute values with one numeric value and that way assign it a vertex in space; this approach creates new problems, though. Algorithm does not measure distances between conceptual values. Let's take, for example, that attribute values are: mammals, fish, reptiles, birds and insects; computers do not understand that the chimpanzees resemble to people and not to bats, and that yoghurt resembles more to an ice cream than a cake. These differences should be assigned higher weights than those that exist between eye colors of different species; hence, algorithm will never be completely accurate since it does not understand what we find more similar in given context. There are four factors that influence the clustering process:

- *Ranks*, arrange values into ascending or descending order, but it gives no information on relative object distance. For example, Marko can be the oldest in the group, and Darko second oldest, but there is no information on that how much Marko is older than Darko.
- *Intervals* show distance between two dimensions. If, for example, Marko is 21 years old, and Darko 19, then interval value shows that Marko is two years older than Darko.
- *Measures* are different from rankings and intervals in the fact that they show absolute values. People should be cautious not to mix intervals and measures, though it seems rather logical, since it can lead to wrong conclusions. For example, if Marija's weight is 60 kg and Olivera's 40 it does not mean that Olivera is 1.5 times slimmer than Marija.
- *Categories*; similar objects are grouped into same categories. CDs can, for example, be computer and music CDs, and latter can again be categorized based on genre and artist. Differences between categories can not be measured nor assigned any values. Actually, it is not possible to say that some software CD is better than a CD with popular music, for example.

There are many different ways to determine object *closeness* or degree of similarity. Methods for object closeness determination are: distance between points in space, record overlapping and angle between vectors in space.

- *Measuring distance between points* in space is the most common way of determining similarity. All attributes of the case are assigned numeric values that represent their coordinates in space along one or more axes. Usually Euclidean distance is measured (reference section 3.2.2).
- *Measuring angle between vectors* that represent objects in space is used to determine similarities between objects that are not obvious on the first glance. Sometimes, the relations that exist between attribute values of the

case are used as a value for that case. Actually, when it is necessary to know relationship between cases and their attributes, it is much more efficient to measure the angle between vectors that start in coordinate centre and end in the point, whose coordinates are case attribute values then to measure distance between those points in space.

- When the cases are consisted of mainly categorical variables, the best option always seeks similarity and *groups cases based on the number of same attributes*. The process is, off course, a bit more complex if there are a number of fields, which cause that the differences in attribute values become less significant. As the number of differences decreases this process becomes more complex. To stress this difference some of the attributes are assigned higher weights. If, for example, attribute that shows the height of a person has four possible values: `short`, `average`, `tall` and `giant` it may happen that a large number of people, with a broad range of heights fall into an `average` category, since the attribute height is not represented in a suitable way. This situation makes the attribute height loose its significance. One possible solution to this problem is to map attribute values into broader classification by multiplying each height by 10, so that the differences between them become visible, i.e. the real 7cm difference becomes 0.7m.

The majority of clustering must load all the data points into memory, which can cause serious scalability problems when processing a large dataset. *Microsoft Clustering* algorithm uses a scalable framework, which selectively stores important portions of the database and summarizes other portions. The basic idea is to load data into memory buffers in chunks, and based on the updated *data mining* model, to summarize cases, that are close together as Gaussian distribution, thereby compressing those cases. When applying this method algorithm needs to scan the raw data only once.

3.3.2 Text mining

Text mining and *data mining* are parts of one much broader field, known as *information mining*. The basic difference between these two fields is in the type of information mined. Eventhough textual information have some implicit structure, basically they are unstructured.

The significance of this field rapidly grows, since the enormous amount of knowledge lays in textual documents. The accessibility of those documents increased drastically with the appearance of web. Hence, the constant growth of amount, and alternation of web text features, requires further research. Unlike the techniques for information access and retrieval, that help users satisfy their information needs, the goals of *text mining* are detection, discovery, and derivation of new information from large collections of textual documents [CHAN01]. In order to discover new knowledge the relations that exist among texts in the collection are detected and examined. One of the basic goals of *text mining* is association rule discovery, trend and event detection.

3.3.2.1 ASSOCIATION RULES

Similar to the association rules discovery in *data mining*, objective measures are support and confidence. The support for the rule *X Þ Y* indicates the number of documents in the collection that contain both, *X* and *Y*. The confidence for the rule denotes the percentage of documents in the collection, which when contain *X* they contain *Y*, too. Association rules are derived from the text collections by following given algorithm:

Let *W'={*$w_1,w_2,\dots,w_m$*}* specify a set of keywords in the document collection *T={*$t_1,t_2,\dots,t_n$*}*. The fact that each t_i is associated with a subset *W'* can be represented with t_i*(W')*. Let *WÍW'*, then the set of all documents *t* in *T* that conform to *WÍt(W')*, is the covering set for *W* and is referred to as *[W]*. In other words *[W]* is the set of all documents from *T* that conform to the rule that each document from *[W]* contains all words from *W* as a part of it's keyword set. Thus, if *WÍW'* then it certainly holds that *[W]Ê[W']*.

Let *R: W=>w* be an associative rule given as implication, where *WÍW'* and *w Î W'-W*. The support *S* for the rule *R* in the collection *T* is defined by the equation *S(R,T)=|[WÈ{w}]|*, which denotes that *S* is an exact number of indexed documents in *T* that contain keywords from *WÈ{w}*. The confidence *C* for the rule *R* in the collection *T* is defined by: $C(R,T)=\frac{|[W\,Y\{w\}]|}{|[W]|}$, and is equal to the conditional probability that the text is indexed by the keyword *w*, if it is indexed by the set of keywords *W*. For the association rule *R* discovered in the collection *T*, it is said that it holds the support ***d*** and the confidence ***a*** if the statements *S(R,t)*^{3}***d*** and *C(R,T)*^{3}***a*** are true, and it is represented with *W Þ w S(R,T)/C(R,T)*.

3.3.2.2 TREND DISCOVERY

Trend is a constant demonstration of the certain pattern over time. The detection of sequential patterns is used for trend discovery in the database transactions. In the document collections trend can be identified by observing the temporal changes in the frequency of appearance of the specific phrases.

Let *W* be the set of all keywords, *P* – the set of all phrases, and *F* – the set of all fields in the document collection *T*, where each specific word is represented by *(*w_i*)*, where w_i*ÎW*; the phrase is represented by $p_i=(w_1,w_2,\dots,w_k)$ (or $<(w_1)(w_2)\dots(w_k)>$), p_i*ÎP*, which actually means that it is one word sequence; and the field is represented by f_i, where f_i *Í WÈ P*, and textual field is a set of words and/or phrases. Then all the documents in the collection *T={*$t_1,t_2,\dots,t_n$*}* can be represented as t_i*={*$TS_i, f_1,f_2,\dots,f_m$*}*, where TS_i indicates the time when the document t_i

was inserted into database (timestamp) and f_i Î F. Algorithm for trend discovery includes three major steps:

- Identification of frequent phrases,
- Generation of phrase history,
- The discovery of patterns that resemble to the actual trend.

3.3.2.3 EVENT DETECTION

Event detection is identification of data in the data array that correspond to new or earlier unidentified events. There are:

- *Retrospective detection,* which refers to the process of identification of earlier unidentified events from the cumulative data collection. Usually, two different techniques are utilized: detection of rapid changes in the term distribution over time and using lexical similarities and temporal closeness of the text documents.
- *On-line detection* identifies new events from the arriving data. The majority of the algorithms are based on the threshold models, which use the following parameters: detection threshold, grouping threshold and frame length.

3.3.3 Web Mining

Web mining is a process of application knowledge discovery techniques for pattern recognition in the data that originates from web. It includes: *web content mining*, *web usage mining* and *web structure mining*.

3.3.3.1 WEB CONTENT MINING

Web content mining is a process of automated pattern discovery in the document contents on web. During this process textual and graphic contents on the web are analyzed. Analysis of textual contents on the web is very similar to the text mining process (refer to section 3.3.2).

3.3.3.2 WEB USAGE MINING

Web usage mining is a process of automated usage pattern detection, from huge collections of web server access logs, which are constantly generated by web server. By analyzing this data access habits of web service users can be discovered and they can be used for improving: web site structure, ways of product advertising and marketing decisions. Data can be collected from various sources including the web server itself, proxy servers and the clients.

3.3.3.3 WEB STRUCTURE MINING

Web structure mining is a process of automated detection of hypertext structure model. This is a process of analyzing structured data used for description of web contents. Structured information can have inter-page or intra-page structure. Information on inter-page structure can be analyzed by following hyperlinks. For representation of this information graph structure is commonly used, where nodes symbolize pages, and branches hyperlinks.

3.3.3.4 DATA PREPARATION FOR WEB MINING

Data has to be prepared for the analysis, which can use a number of different techniques: statistical analysis, association rule discovery, clustering, classification or detection of sequential patterns. Data cleaning process is site-specific and involves tasks such as merging log files from multiple servers, removing graphics file accesses, and parsing of the logs. For further use in the personalization process it is necessary to identify the user and the set of *user sessions* (refer to section 3.1.1.2). For web site using *cookies* or embedded session IDs, or are situated on *Java -enabled* web servers this identification is trivial.

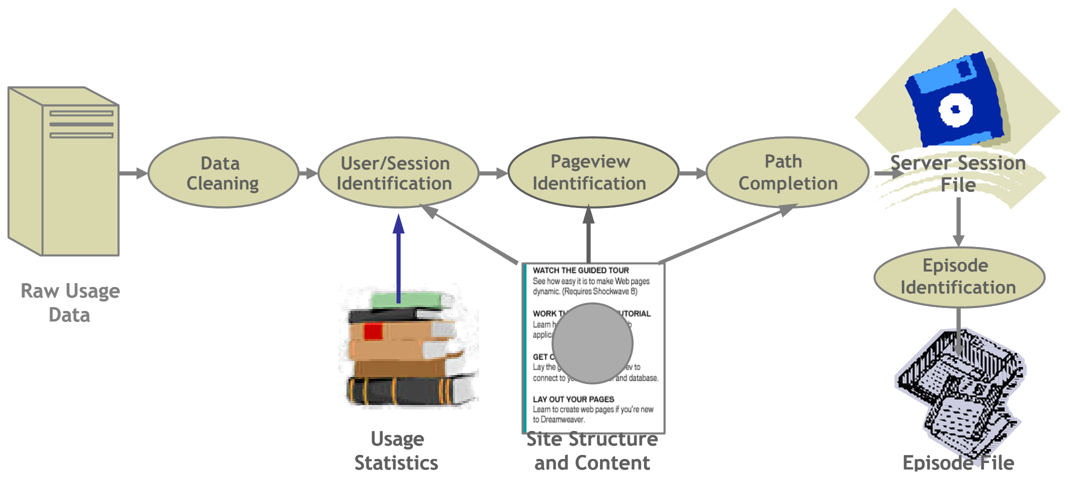


Figure 3.7 Summary of preprocessing steps

Since the HTTP is a stateless protocol, as a result of one user action several file requests are executed (HTML, images, audio etc.) Set of page files that server returns as a response to a single user action represents a *pageview*. After session identification log file has to be cleaned or transformed into the list of relevant *pageview*. Cleaning the server log files involves removal of all redundant information on file access and leaving just one entry for one *pageview*. During this step site structure and content are processed. All, the pages that contain frames and dynamically generated pages (that use the same template name for more than one *pageview*) are handled. It is necessary to check whether all extraneous references (such as image or sound files) are correct, since the recommendation engine should not provide dynamic links to "out-of-date" or non-existing pages (*web structure mining*). Content preprocessing consists of converting the text, image, scripts, and other multimedia files into forms, which can be used in the *web usage mining* process. Usually, this consists of performing *content mining* such as classification or clustering which gives results that can later be used to limit the discovered patterns to those containing pageviews about certain subject or class of products.

One of the consequences of caching on both client side and proxy server is that the data in log files is usually incomplete; hence, several simple heuristic methods for *path completition* are obtained (detailed description can be found in [COOL99]). Dynamic contents with unique URIs for each server session cannot be cached on proxy level; however, any type of content can be cached at the client level, where the amount of caching is set by the client-side browser.

Each user session in the user session file can be viewed in two ways:

- As one transaction (*episode*) that contains several *pageviews,* or
- As a set of more transactions, where each of them consists of one page reference.

The goal of episode identification is a dynamic creation of meaningful clusters of references for each user, based on an underlying model of user's browsing behavior (for more detailed description, refer to [COOL99]).

Finally, the session file can be filtered by removing irrelevant transactions and low-support URI references (URIs that do not appear in a sufficient number of sessions). This type of support filtering can be useful in eliminating noise from the data, and it can also be understood as a type of dimensionality reduction for clustering algorithms that use URIs, appearing in the session file, as features.

3.4 Sample academic recommender systems

Even though the beginning of a commercial boom was in the late nineties the idea on personalized, user-adaptive systems appeared much earlier (refer to [KOB01]). In these first personalized systems, the user modeling was performed by the application system, and often no clear distinction could be made between system components that served user modeling purposes and components that performed other task. In the end of 1990s, the value of web personalization was increasingly recognized in the area of electronic commerce. Web personalization allows product offerings, sales promotions, ad banners, etc. to be targeted to each individual user. The relationship with users on the Internet migrates from anonymous *mass marketing* and sales to 1-to-1 marketing. The most important characteristic of most contemporary systems is their client-server architecture. User modeling systems are not functionally integrated into the application, but communicate with the application through inter-process communication and can serve more than one user (client) applications at the same time. Client-server architecture provides a number of advantages compared to *embedded* user modeling components [KOB01]:

- Information about the user is stored in a central or virtually integrated repository and put to the disposal of more than one application at the same time.
- Information on user acquired by one application can be employed by other applications.
- Information about users is maintained in non-redundant manner. This way the consistency and coherence of information gathered by different applications can be achieved easier.
- Information on user groups, either available as stereotypes or dynamically calculated as user group models can be maintained with low redundancy.
- It is possible to apply different methods and tools for system security, identification, authentication, access control and encryption for protecting user models in user modeling servers. There are a lot of company privacy policies, industry privacy norms and conventions, national and international privacy legislation, and privacy-supporting software tools and service providers.
- Additional user information that is dispersed across the organization (e.g. demographic data from client databases, past purchase data from transactional systems, user clustering from marketing research) can be integrated more easily with the information in the user model repository. To access external data, ODBC interfaces or native support for a wide variety of databases are a must. Due to legacy business processes and software, external user-related information often continues to be updated in parallel to the *e-commerce* application and therefore needs to be continually integrated at reasonable costs and without impairing the response time.

However, these systems also have some disadvantages; for example, necessity of a network connection and potential line of failure.

The last three sections contain detailed descriptions of all contemporary types of personalization and some of the methods for knowledge discovery from existing and data collected with the goal of automated recommendation generation, as "the most intelligent" means for the personalization. The most intelligent in a sense that proportionally from the minimum input data, it produces high quality adaptations. Therefore, a number of experimental and commercial systems implement this type of personalization. In the remainder of this chapter some of them are described.

3.4.1 WebWatcher

WebWatcher [JOA97] is one of the first recommender systems. It appeared as an "intelligent" response to keyword-based search engines and represents a typical content based system. It was developed on Carnegie Mellone University in Pittsburgh, 1995.

WebWatcher acts as a *learning apprentice* observing and learning from users' actions. Over time ***WebWatcher*** learns to acquire *greater expertise* for the parts of the web that it has visited in the past, and for the types of topics in which previous visitors have had an interest. ***WebWatcher*** is implemented as a server, on a separate workstation on the network and acts similar to proxy. Before returning a page to a user it makes three modifications: adds the ***WebWatcher*** command list on the top of the page; each hyperlink in the original page is replaced by a new URI that points back to the ***WebWatcher*** server; if ***WebWatcher*** calculates that any of the links on this page is strongly recommended by its search control knowledge, then it highlights them to suggest them to the user.

While it waits for the user's next step, ***WebWatcher*** pre-fetches web pages it has just recommended to the user to minimize network delays. Figure 3.8 illustrates one cycle of user interaction. When the user clicks on a new hyperlink, ***WebWatcher*** updates the log for this search, retrieves the page (unless it has already been prefetched), performs similar substitutions, and returns the copy to the user. This process continues until the user elects to dismiss the agent. ***WebWatcher*** has the task to suggest an appropriate link given user interest and actual web page. In other words, it requires knowledge of the following target function:

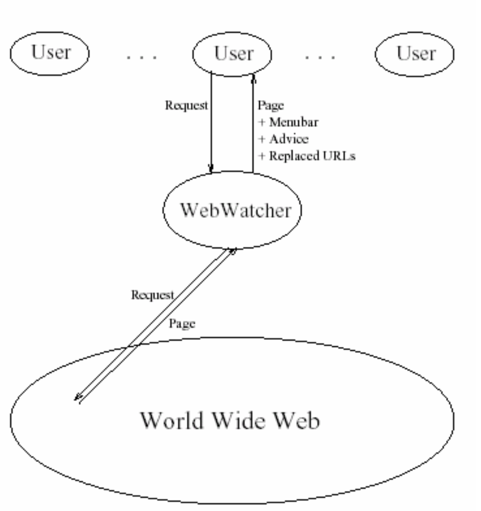


Figure 3.8 WebWatcher is an interface agent between the user and WWW

LinkQuality : Page x Interest x Link-> [0,1]

The value of function *LinkQuality* can be interpreted as the probability that the user will select *Link* given the current *Page*, and knowing user's *Interest*. The three different approaches to learning this target function from experience are examined: learning from the previous tours, reinforcement learning based on the hypertext structure, and the method combination of the first two approaches.

3.4.1.1 LEARNING FROM PREVIOUS TOURS

This approach learns by annotating each hyperlink with the interests of the users who took this hyperlink on previous tours. Thus, whenever a user follows a hyperlink the description of this hyperlink is enlarged by adding the keywords the user entered at the beginning of the tour. During an active tour ***WebWatcher*** compares the current user's interests with the descriptions of all hyperlinks on the current page, and suggests those hyperlinks which have a description sufficiently similar to the user's interests. Both the user interests and hyperlink descriptions are represented using the TFIDF model, and their similarity is calculated as a cosine between vectors (refer to sections 3.2.1 and 3.2.2).

3.4.1.2 Learning from Hypertext Structure

In the learning method that learns from the previous tours each hyperlink is represented with the stated interests of earlier users who selected it. This method augments each hyperlink using the words encountered in the pages downstream of it. This method is based on reinforcement learning technique. The objective is to find paths through the web, which maximize the amount of relevant information encountered ie. to allow agents learn control strategies that select optimal actions in certain settings.

In this section one possible application of this algorithm is illustrated, while the detailed description can be found in [JOA97]. Web pages represent all possible states, and hyperlinks actions, that the agent can take. Suppose that a web agent looking for pages on which some word *w* (for example, *intelligent*), navigates from one state to the other by performing actions i.e. following hyperlinks. At each state *s* the agent receives a certain reward $R_w(s)$, which is equal to the TFIDF value for the word *w* on page *s*. The goodness of an action *a* can be expressed of an evaluation function *Q(s,a)* defined for all possible state-action pairs. The value of this function *s* and hyperlink *a* is discounted sum of future rewards (TFIDF values for the word *w*) over the optimal tour beginning with *a*.

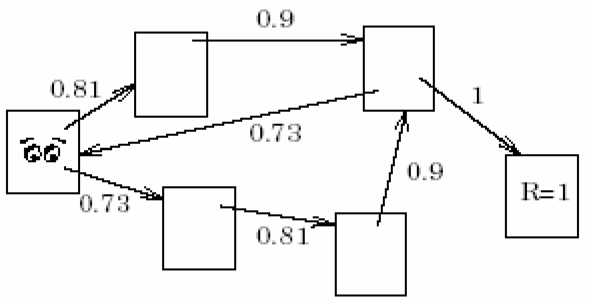


Figure 3.9 Example state space

The example in figure 3.9 is used to illustrate the algorithm. Nodes represent web pages, and edges hyperlinks. Each hyperlink is annotated with value of the function *Q(s,a)*; that state gives reward 1, while there is no reward in other states. If the agent always follows the action with the highest *Q* value, it will get to the reward state in the smallest number of steps and thus maximize the discounted reward it receives (actually, follows the optimal path).

3.4.1.3 Recommendation of New Contents

Because ***WebWatcher*** cannot expect that users will always stick to the pages it has already seen, a core question in implementing this approach is how to learn a general approximation for each of the *Q*-function to that applies even to unseen pages and hyperlinks. Each hyperlink *a* is described by the TFIDF vector representation of the underlined anchor text, each page analogously by its title. New contents are recommended based on their similarity with those already examined. In fact, hyperlink a_1 on page s_1 is similar to hyperlink a_2 on page s_2 if the value of the function $a_1 \cdot a_2 + 2 \cdot s_1 \cdot s_2$ is minimal.

3.4.1.4 PERSONAL WEBWATCHER (PWW)

Whereas ***WebWatcher*** learns to specialize to a specific web locale, ***personal WebWatcher (PWW)*** [MLA96] learns to specialize to a particular user. Analogous to ***WebWatcher***, ***PWW*** observes user actions, but it does not include the user into the learning process (does not require him to enter keywords, nor comment texts read). This agent learns a model of long-term users' interests by observing which pages they do and do not visit. It stores followed hyperlinks, and recommends the user those that it considers would be interesting for that specific user. During the learning phase (usually over night) agent analyzes stored data and updates model of user interests.

3.4.2 WebPersonalizer

System ***WebPersonalizer*** [MOB00] employs the architecture shown on figure 3.10 to provide a list of recommended hyperlinks to a user while browsing a web site. This system calculates recommendations based on hypertext structure of a site and anonymous usage data provided by web server logs. The overall process of usage-based web personalization is divided into two components.

3.4.2.1 OFF-LINE PROCESS

The *off-line* component involves two levels: in the first phase data preparation tasks are executed. During this phase data is prepared for the analysis (refer to section 3.3.3.4); server log files are converted into *server sessions*. Two different *data mining* techniques are used for aggregate usage profiles discovery from these session files. Usage profiles exhibit three important characteristics – they should:

- Capture possibly overlapping interests of users (since many users may have common interests up to a point, in their navigational history, beyond which their interests diverge);
- Provide the capability to differentiate among pageviews in terms of their importance within the profile;
- Have a uniform representation, which allows for the recommendation engine to easily integrate different kinds of profiles (multiple profiles based on different pageview types, or obtained via different mining techniques).

Given these requirements, representation of usage profiles, as weighted collections of URIs (that were visited by the user during session) provides a great deal of flexibility [MOB00]. For usage profile discovery two different methods, based on clustering and association rule discovery are used. After the execution of a *data mining* task, detected *frequent itemsets (FIs)* and URI clusters are used by the on-line component to compute recommendation set.

If the clustering method is used for usage profiles detection the computation of session clusters from identified sessions first has to be accomplished (refer to section 3.3.3.4). Then the detected clusters are used for derivation of aggregate usage profiles. This procedure starts by computation of cluster centroids (the mean vectors). The mean value for each URI in the mean vector is computed by finding the ratio of the number of occurrences of that URI across all sessions to the total number of sessions in the cluster and it represents the weight for that URI.

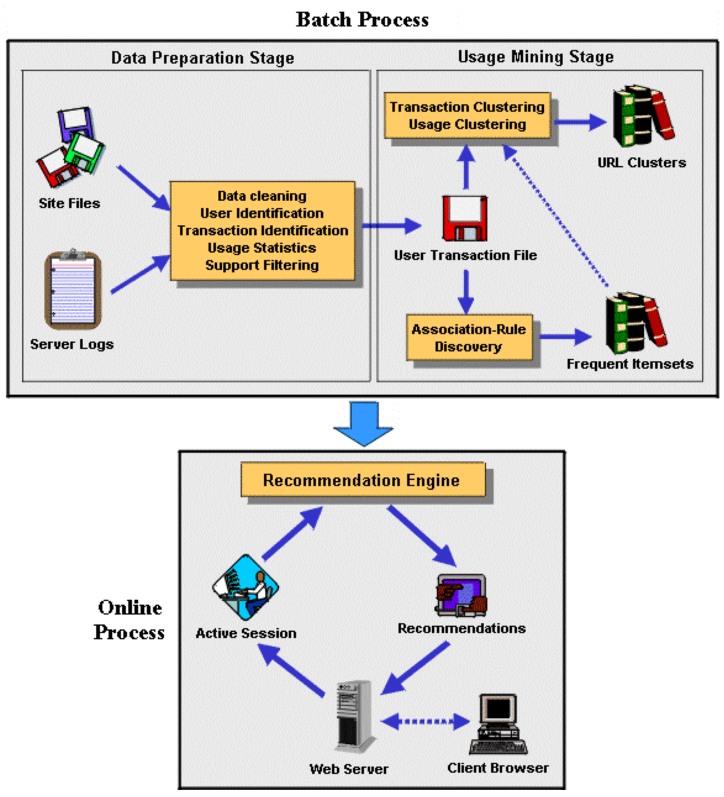


Figure 3.10 Architecture of *web-usage based* personalized system

For the second method, the ***WebPersonalizer*** system uses *Association Rule Hypergraph Partitioning* (ARHP) technique (refer to [HAN97]). This technique is well-suited for the task since it provides automated filtering capabilities, and does not require distance computation. ARHP is, also, very useful technique for clustering high-dimensional data sets because it does not require dimensionality reduction. In the ARHP technique the set of *FIs* are used as hyperedges in a hypergraph (hypergraph is an extension of a graph in the sense that each hyperedge can connect more than two vertices). The weights associated with each hyperedge are computed based on the confidence of the association rules involving the items in the frequent itemset. The hypergraph is recursively partitioned into a set of clusters. Each cluster represents a group of items (URIs) that are very frequently accessed together across sessions. The *connectivity* value of vertex (URI appearing in the frequent itemset) with respect to a cluster measures the percentage of edges with which the vertex is associated. The significance weight of the URI within the profile is obtained as a function of the connectivity value.

3.4.2.2 ON-LINE PROCESS

On-line component comprises recommendation engine and HTTP server. Web server monitors the active server session as the client browser makes HTTP requests. This can be achieved in several ways, among which *URI rewriting* and provisional caching of access logs on the web server. Recommendation engine takes the active user session and compares it with aggregate usage sessions, to determine the recommendation set of URIs. When determining similarity the system normalizes for the size of clusters and the active session (size of the corresponding URI collection). This is very important when matching active sessions with clusters to get recommendation. If, during the active session, two matching clusters have the same value of the unnormalized matching function, larger cluster should be ranked lower. This corresponds to the intuitive notion that we should see more of the user's active session before obtaining a better match with a larger cluster. The recommended objects are then added to the last page in the active session accessed by the user before the page is sent to the browser.

3.4.3 Fab

Fab [BAL97] is a distributed hybrid system, and is a part of the Stanford University digital library project. The process of recommendation in this system can be partitioned into two stages: first the collection of contents to form a manageable database or index, and subsequently selection of contents from this database for particular user. In the collection stage pages relevant to a small number of topics are gathered. These topics represent computer-generated clusters of interests, which track the changing tastes of the user population. In the selection stage collected pages are delivered to the users. One topic can be interesting to many users, and one user can be interested in many topics.

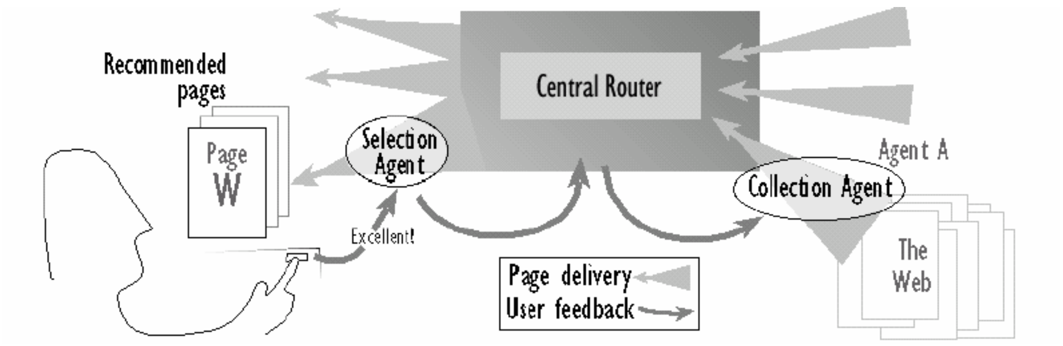


Figure 3.11 Overview of the Fab architecture

An overview of the implemented architecture, consisting of agent community and the central component is illustrated on the figure 3.11. Every agent maintains a profile (represented by the *vector-space* model; section 3.2.1), based on words contained in web pages, which have been rated. System comprises three basic components: *collection agents*, *selection agents* and central router.

3.4.3.1 COLLECTION AGENTS

Collection agents find pages for a specific topic. A collection agent's profile represents its current topic. The population of collection agents as a whole adapts to the population of users, not to any specific user. Unpopular (whose pages are not seen by many users) and unsuccessful collection agents (who receive low median feedback scores) are regularly weeded out and the best ones duplicated to take their places. Thus, the collection agents' specializations need to be fixed in advance, but are determined dynamically and change over time. Several different types of collection agents are implemented [BAL97a]:

- *Search agents* execute *best-first* web search and try to locate pages best matching their profiles. Their assumption in that a page will have links to similar pages, and so by following links from page to page they can uncover information relevant to a particular topic.
- *Index agents* construct queries to pass to various commercial web search engines that have already performed exhaustive indexing.
- *Non-adaptable agents*, can be: agents that supply randomly picked pages (*random agents*); agents that collect various human-picked *'cool sites of the day'* (*cool agents*) and agents that attempt to serve an average user (*no-memory index agents*). The last agent type, rather than maintaining their own specialized profile, has profiles that represent an average of all the user profiles in the system.

It is possible to instantiate a smaller number of collection agents than there are users, perhaps even a fixed number. This should allow the system to scale elegantly as the number of users and documents rise. The exact number of collection agents required is determined by several factors, including the amount of the overlaps between users' interests and the tradeoff between the available computing resources and the quality of recommendations required. These agents automatically identify evolving communities of interest, allowing the support of social interactions between like-minded people and automatically provide group as well as individual recommendations. Effectively, like-minded users are combining their resources, as each collection agent will be receiving feedback from all user interested in a topic.

3.4.3.2 CENTRAL ROUTER

Central router sends pages found by the collection agents, to the users whose profiles they match above some threshold. This way each user receives pages matching their profile from the collection agents.

3.4.3.3 SELECTION AGENTS

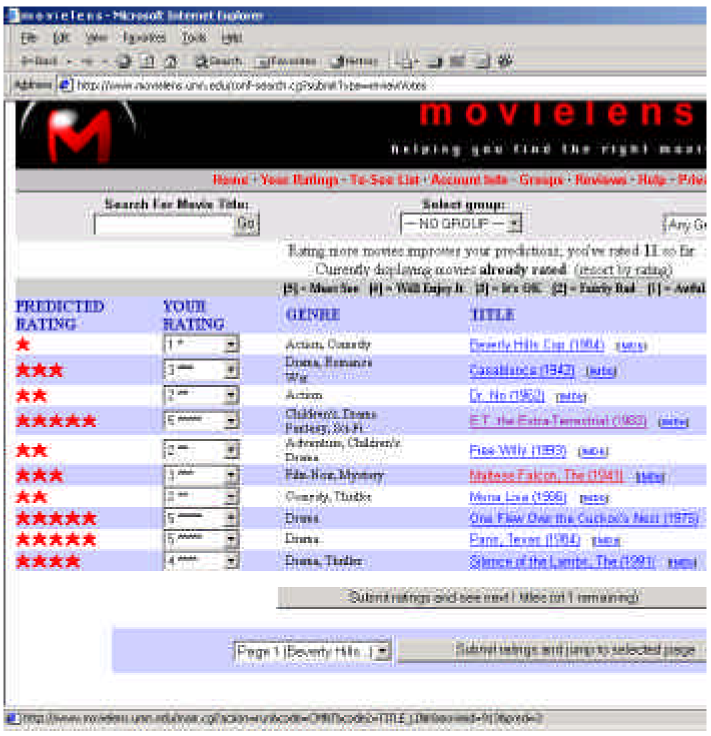
The selection agents from the collected pick pages, which will be interesting to a specific user. The selection agent profile corresponds to the interests of one user. It, also, implements additional functionalities: discards pages that the user has already seen, and insures that in any single batch of recommendations (usually 10 pages) exists at most one page from any site. When the user has requested, received, and looked over their recommendations, they are required to assign appropriate ratings (from a 7 point scale). The user's feedback represents a significant investment in time and effort; hence, they are stored in their own private selection agent's profile, and that way it is insured they can never be covered by other user's feedback. Therefore they are easily exportable for use in other applications. The user's ratings are used to update their personal selection agent's profiles. During this process their values are mapped: interval [1,7] is mapped onto [-3,3], and then used in ***Roccio*** method as correction factor (refer to section 3.2.2). Those ratings are also forwarded back to the originating collection agents, which will use them to adapt their profiles; while any highly rated pages are passed directly to the active user nearest neighbors. The private selection agent of the neighbor processes these recommended pages in the same way as the pages received from the central router.

All the advantages of the hybrid system are exhibited in the selection process: When making collaborative recommendations, others' experience is used, rather than incomplete and imprecise content analysis methods. By making content-based recommendations, it is possible to show items unseen by others or make good recommendations to users, even if there are no other users similar to them. It is feasible to make collaborative recommendations between users who have not rated any of the same items (as long as they have rated similar items). In this way the reach of collaborative systems is extended to include databases, which change quickly or are large with respect to number of users. Using group feedback potentially decreases cycles required to achieve the same level of personalization.

3.4.4 MovieLens Matcher

MovieLens Matcher [HER01] is implemented as an experimental extension for ***MovieLens,*** a movie recommendation web site. ***MovieLens*** is a free service provided by ***GroupLens*** research group from the Minnesota University. ***MovieLens*** was already using interest recommender based on CF technology. Figure 3.11b shows a list of movies that ***MovieLens*** recommends based on the list of rated movies (figure 3.11a). The greatest weakness of this system, and most current CF systems, that generate recommendations based on user ratings is their basis solely on historical ratings data. This approach assumes that a user's interest is independent of the task at hand. In reality, actual task or context greatly affects the value of recommendation, which current systems solve by relying on content-based query engines that use either metadata or full-text indexing and analysis. However, for development of ***MovieLens Matcher*** system a task-focused approach to recommendation that is entirely independent of the type of content involved and based primarily on ratings data is used. Since most of the current systems already use this approach, minimal additional data collection from users is required and no requirement for metadata exists. Figure 3.12 illustrates the architecture of this system.

a)



movielens - Microsoft Internet Explorer

movielens — helping you find the right movi

Home · Your Ratings · To-See List · Account Info · Groups · Reviews · Help · Priv

Search For Movie Title: [Go] Select group: -- NO GROUP -- Any G

Rating more movies improves your predictions; you've rated 11 so far
Currently displaying movies **already rated** (resort by rating)

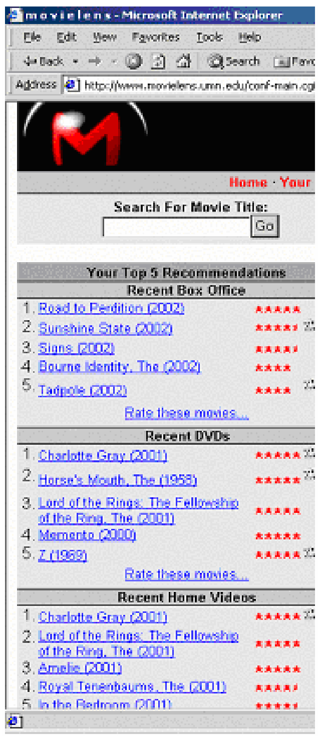
[5] = Must See [4] = Will Enjoy It [3] = It's OK [2] = Fairly Bad [1] = Awful

PREDICTED RATING	YOUR RATING	GENRE	TITLE
★	1 *	Action, Comedy	Beverly Hills Cop (1984) (info)
★★★	3 ***	Drama, Romance War	Casablanca (1942) (info)
★★	2 **	Action	Dr. No (1962) (info)
★★★★★	5 *****	Children's, Drama Fantasy, Sci-Fi	E.T. the Extra-Terrestrial (1982) (info)
★★	2 **	Adventure, Children's Drama	Free Willy (1993) (info)
★★★	3 ***	Film-Noir, Mystery	Maltese Falcon, The (1941) (info)
★★	2 **	Comedy, Thriller	Mona Lisa (1986) (info)
★★★★★	5 *****	Drama	One Flew Over the Cuckoo's Nest (1975)
★★★★★	5 *****	Drama	Paris, Texas (1984) (info)
★★★★	4 ****	Drama, Thriller	Silence of the Lambs, The (1991) (info)

Submit ratings and see next 1 titles (of 1 remaining)

Page 1 (Beverly Hills ... | Submit ratings and jump to selected page

b)



movielens - Microsoft Internet Explorer

Address http://www.movielens.umn.edu/conf-main.cgi

Home · Your

Search For Movie Title: [Go]

Your Top 5 Recommendations

Recent Box Office

1. Road to Perdition (2002) ★★★★★
2. Sunshine State (2002) ★★★★½
3. Signs (2002) ★★★★½
4. Bourne Identity, The (2002) ★★★★
5. Tadpole (2002) ★★★★

Rate these movies...

Recent DVDs

1. Charlotte Gray (2001) ★★★★★
2. Horse's Mouth, The (1958) ★★★★★
3. Lord of the Rings: The Fellowship of the Ring, The (2001) ★★★★★
4. Memento (2000) ★★★★★
5. Z (1969) ★★★★★

Rate these movies...

Recent Home Videos

1. Charlotte Gray (2001) ★★★★★
2. Lord of the Rings: The Fellowship of the Ring, The (2001) ★★★★★
3. Amelie (2001) ★★★★★
4. Royal Tenenbaums, The (2001) ★★★★½
5. In the Bedroom (2001) ★★★★½

Figure 3.12 List of rated (a) and recommended movies (b)

3.4.4.1 INTEREST RATINGS

A collection of numeric ratings, where each rating indicates user's interest in a specific item. Ratings can be any scale, but are generally discrete (common ranges are 1 to 10, 1 to 7, 1 to 5, and 0 to 1).

3.4.4.2 INTEREST RECOMMENDER

In a traditional recommender system interest recommender is the component that predicts ratings from historical ratings database. This component uses classic CF approach. It provides an external interface that allows other processes to request an interest prediction for any *(user, item)* couple. It cannot produce task focused recommendations by itself.

3.4.4.3 TASK SPECIFICATION

Task specification consists of input data that indicates what kind of task the user has in mind to complete. It contains a list of example items associated with actual task, known as a *task profile*. There are two distinct approaches to collecting task profiles. In the first approach user must explicitly specify items associated with the task. For example, if the goal is to find gift recommendation for children from a book recommender like ***Amazon.com***, it is possible to select two or three items that that children already own and like. The benefit of this approach is that the items in the task profile are guaranteed to be associated with the user's task. The drawback is that the user must do the work. The task-focused recommendations must therefore justify the additional work by offering notably more value than general-interest recommendations. The second approach is automatic. The system observes user behaviors, such as items purchased or a web page visited, and infers a task profile. If, for example, a user places a hammer into a shopping basket, the system could use it as a single item task profile. From this profile, it might recommend nails. Task profiles created this way do not require additional work from the user, but they can be false and thus require recommendation algorithms that find task associated items even when errors exist in the task profile.

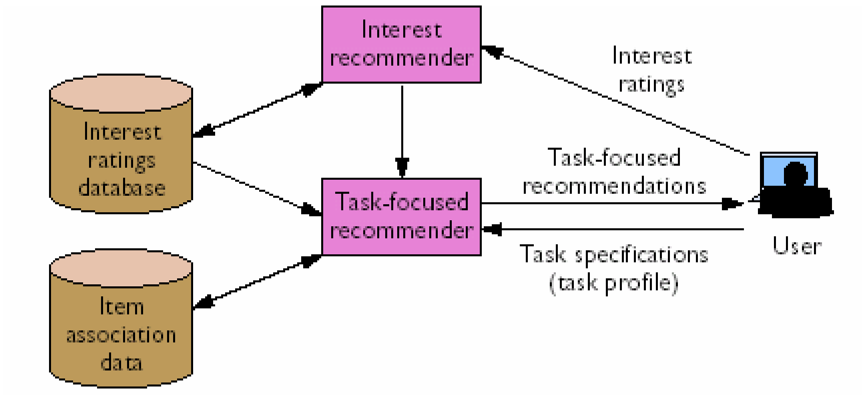

Interest recommender
Interest ratings
Interest ratings database
Task-focused recommendations
Task-focused recommender
Task specifications (task profile)
User
Item association data

Figure 3.13 Task-focused recommender system architecture

3.4.4.4 ITEM ASSOCIATION DATA

These are descriptions of associations between items available for recommendation. This data is automatically generated by the task–focused recommender via analysis of the interest ratings data. Given a task profile, the problem of task-focused recommendation reduces to one of identifying associations between items. Each item that is associated with all items in a task profile is likely to be associated to the user's task. Primary mechanism for discovering item associations automatically, independent of content, uses the existing user interest-ratings data. This data can be represented as a matrix, where each row represents a user and each column represents an item. Recommendation engines that use CF approach determine the similarity between users by correlating the rows to find the rows of ratings that agree. For task-focused retrieval, however, associations between items should be identified, and not users. Thus, the task-focused recommender computes correlations between columns.

A strong positive correlation between items *i* and *j*, in general, means that the higher the rating a user gives to *i*, the higher the rating to *j*. Hypothesis is that correlation will capture associations between items. If a single user, for example, gives the same rating to two items, then there is a probability that those items are associated or contain common elements. As the number of users who give those two items an identical or similar rating increases, the probability that those items have something in common increases. If the correlations over the entire user base are computed, a high positive correlation could represent significant common elements. In the majority of *e-commerce* systems a set of items is relatively static compared to the number of users that changes most often [SAR01]. Thus, pre-computing the all-to-all similarity significantly cuts down the time necessary for recommendation generation and improves the system performance. Item associations are computed in a nightly *batch* process. To reduce the amount of storage, only the top 100 most-correlated associations for each item are kept.

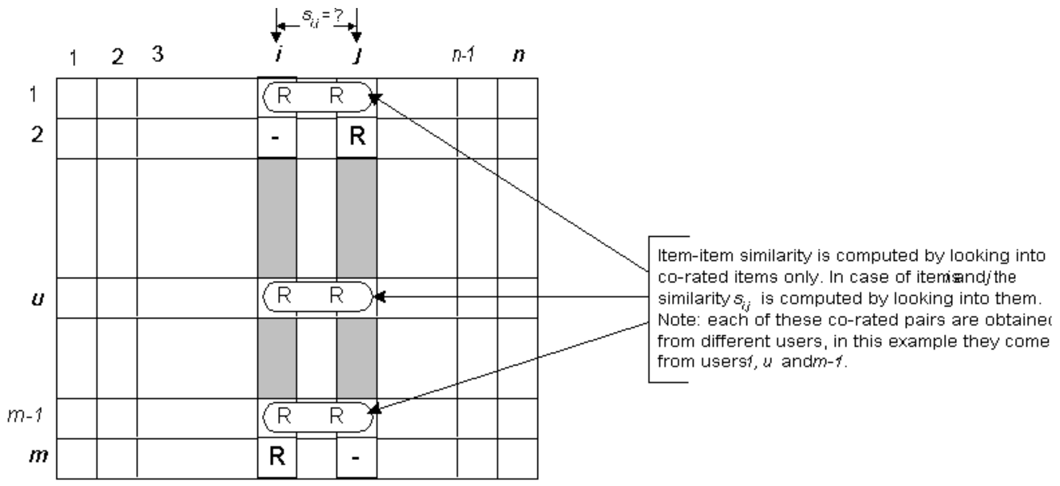

$s_{i,j} = ?$
1
2
3
i
j
n-1
n
1
2
u
m-1
m
R R
- R
R R
R R
R -
Item-item similarity is computed by looking into co-rated items only. In case of items i and j the similarity $s_{i,j}$ is computed by looking into them. Note: each of these co-rated pairs are obtained from different users, in this example they come from users 1, u and m-1.

Figure 3.14 Isolation of simultaneously rated items and similarity computation

3.4.4.5 TASK-FOCUSED RECOMMENDER

The task-focused recommender is the component that gathers a task description from the user and returns a set of appropriate recommendations. In this approach, the task-focused recommender operates on rating data, predictions from the interest recommender, and a task profile. It computes both the item associations and the task-focused recommendations. Figure 3.14 illustrates *on-line* process of computing task-focused recommendations.

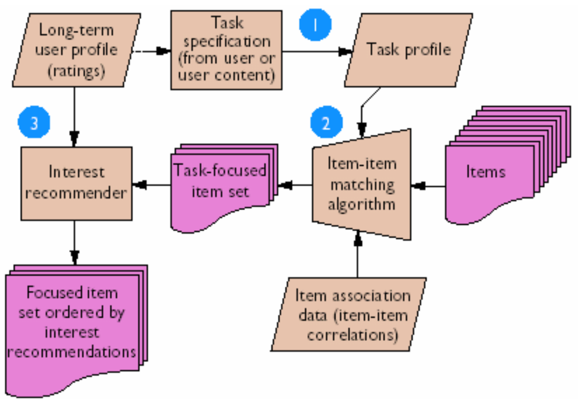

Long-term user profile (ratings)
Task specification (from user or user content)
1
Task profile
3
2
Interest recommender
Task-focused item set
Item-item matching algorithm
Items
Focused item set ordered by interest recommendations
Item association data (item-item correlations)

Figure 3.15 Data flow diagram

First (❶), the user specifies a task profile to the task-focused recommender. (the dotted line indicates that users often specify previously rated items). Items in a task profile are referred to *query items*. Then (❷), the task recommender uses the precomputed item associations to identify those items most likely to be associated with the items in the task profile. The result is a *task-focused item set*. To create a task-focused item set, only database items that occur in top correlates list for each query item are selected. Finally (❸), the interest recommender re-ranks the set based on the interest predictions and returns the resulting ordered recommendation list to the user. Items that occur close to the top of the lists of all query items are more likely to be selected. The actual value of the correlation used to identify the association is not considered, only the ranking. If the ratings database is sparse, that is, if each user rates only a small percentage of all items, then certain *(item, item)* pairs will have a very little overlap. For example, for two movies, A and B, there may be only a small number of users who have rated both A and B. Correlations based on small number of data points may be misleadingly high and incorrect. To solve this problem, a technique common in *data mining*, namely, *support* is used (refer to section 3.3.1.2). Hence, only those correlations that have the support above a specified support threshold are considered.

3.4.4.6 TASK-FOCUSED RECOMMENDATIONS

Task-focused recommendations consist of a ranked list of items that are most probably associated with the current task while also being recommended by other users with similar interests.

The interest ratings and recommender components of this architecture exist in any current CF system for predicting interests; the other four components are unique to this solution.

3.4.5 Review

In this section just some of the numerous examples of academic recommender systems are illustrated. They, all use different variations and combinations of previously explained techniques for recommendation generation. Table 3.2 contains a comparative assessment of their features.

Table 3.2 Comparative assessment of several academic systems

	usage	*approach*	*user profiles*	*similarity computation*	*user engagement*
WebWatcher [95]	a web site	IR	Short-term, anonymous (TFIDF)	cos-based	Explicit (indicates interests) / implicit (follows links)
Fab [96]	indicated topics	hybrid (IR/CF)	Long-term (TFIDF combined with ratings)	cos-based	Explicit (interests and ratings ranging 1-7)
WebPersonalizer [00]	a web site	CF	Long-term, anonymous (URI lists)	cos-based, ARHP	Implicit (follows links)
MovieLens Matcher [01]	indicated task	CF	Long-term (rated contents) and Short-term (task profiles)	cos-based, Pearson and adjusted cos-based	Explicit (ratings from 1-5 and task specification)

4 Portal Framework

4.1 Infostructure

One of the main objectives of ***ETH World*** project consists in improving the virtual information infrastructure, so called *infostructure*. In such manner, information becomes available to all potential participants: ETH staff, students, alumni,… anytime and anyplace.

It is evident that this virtual infrastructure is very heterogeneous from the technical point of view. Information must be available on various devices, while using different access methods. Users are situated in the spatially distributed organizations and are specialized in different technical domains. Infrastructure of the ***ETH World*** project has to provide each user with the system that satisfies his needs.

The ***ETH World*** portal [JAUS01], for the user, represents a specific point of entry into the virtual environment. As such it represents central interface element between the infrastructure and the user. Infrastructure is consisted of services (applications), which are available to the user through the portal. In this sense, the portal has two facets: towards the applications it behaves as a run-time environment, while for the user it is information presenter. For this reason, while deploying the ***ETH World*** portal prototype some of the main issues were: personalization, security, dynamic representation, price, scalability and complexity of maintenance and deployment.

4.1.1 Personalization

Various users have various requests from the system. Everyone has preferences of his/her own and thus adapts the environment to own habits. Hence, the portal can provide the user with methods, which will, for example, enable user to, by setting particular parameters (*settings*), adapt running the existing applications to his/her needs. Parameters should be persistently stored, so that user can access them anytime, any place.

As explained (section 2.5.10), the basis for dealing with personalized web applications is user identification. After the user enters his credentials (user name, password) that authentication stays valid for the rest of that session. Figure 4.1 illustrates security scheme applied in this prototype.

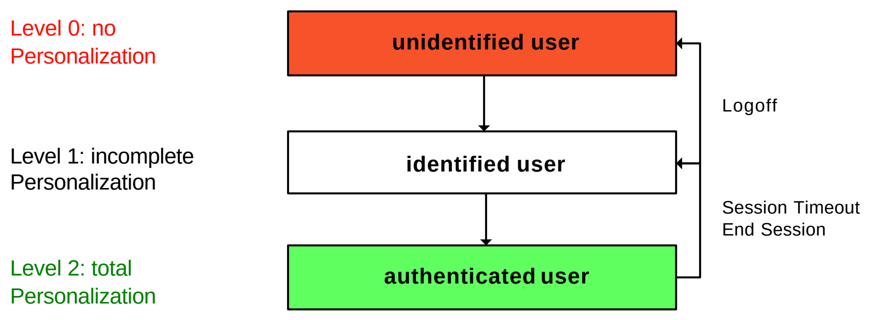

Level 0: no Personalization
unidentified user
Logoff
Level 1: incomplete Personalization
identified user
Session Timeout
End Session
Level 2: total Personalization
authenticated user

Figure 4.1 Identification scheme used in the prototype

User can change the system behavior. Thus, user can turn off the identification. If some security scheme is applied, portal utilization becomes more complex, since each session requires authentication. User can, either identify himself to each portal application, once during one session, or utilize single sign-on mechanism. After the successful user identification, it is possible to set all personalized parameters in the application. Security checks for particular parameters are executed by the application. If the further refinement of access rights is necessary, application can generate its own *resource ids*, and check them in runtime.

4.1.1.1 SITES AND SESSIONS

The foundation for every personalized application is that the user does not have to present him/herself to each site, (s)he visits during one session. Since all the client-server communication goes through HTTP protocol, where each page is considered to be a response to one user request; but which by being stateless does not enable web server to easily find connection between particular requests. Saving status information on the server-side can be accomplished in several ways. One of them is forwarding the parameters between particular pages. However, this significantly increases the amount of the transferred data. For that reason, most often *user sessions* are administered. Session is defined as one access to a site in a specified time period. It is identified using *session ID* that is forwarded as a parameter. RFC 2109 [RFC] describes other mechanism, known as *Cookie*. This mechanism is implemented by all web browsers and represents elegant alternative to parameter method; for that reason it is used in this prototype, too.

4.1.1.2 USER GROUPS

In the ETH working environment users can be assigned to various groups. For example, one assistant on the Computer Science Department in his/her work uses the portal different from one assistant on the Chemistry Department, and this one again, completely different from any student or member of the administrative staff. Each of them has the experiences of his/her own, own tasks and preferences. Hence, there is one thing they all share; they are all part of ETH. Figure 4.2 illustrates existing organization on the ETH, which has to be supported by the postal. The ETH University is consisted of multiple technical departments: Computer Science (D-INFK), Electrical (D-ELEK), Chemistry (D-CHEM) and so on. Each of these departments comprises several institutes. For example, computer Science Department includes Institute for Information Systems, Institute for Theoretical Informatics… and each of them employs a number of professors and assistants. Furthermore, on each of the departments a number of students attend lectures, sit exams …

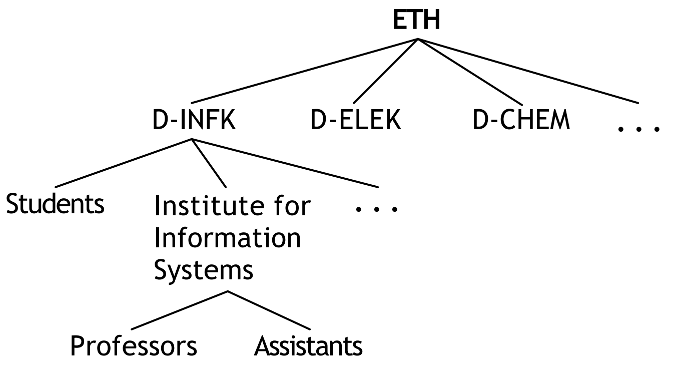


Figure 4.2 Implemented group hierarchy

Through the established group hierarchy particular applications and information access rights are inherited, as well as specific parameters. In such manner, for example, ETH font becomes available to all portal users, since it has been set as a parameter of the top level hierarchy group. If they want to, Institute for Information Systems employees can change this font for whole institute, and each specific user can set his own font. In some situations default parameter value for the group, on the high position in the hierarchy, must not be personalized. Thus, there has to exist the possibility to set these parameters as fixed. Then changing their values by the lower levels in the hierarchy is termed *overriding*. In the existing model each user can be the member of only one group; this works for the groups too. However, it is possible that, for example, graduated student is at the same time a PhD student, which means that (s)he'd belong to two different groups. This problem is possible to solve in the following manner: user can hold several different roles (*Personalities*). Or, two accounts can be used. Thus, in the current portal version this is not practicable.

4.1.1.3 SETTINGS

Each application can generate new, or change existing parameter values for specific users and user groups. The application does not have to know anything about available memory and access methods. Parameters can be accessed by using key values. This method resembles to ***Windows Registry***. However, one should be aware that possible sets of s parameter values should be small. Managing large amounts of binary data (BLOB) is taken over by the application.

4.1.1.4 MODULARITY

As already mentioned the majority of portals comprise of the home page that can be customized and specific parts, so called *modules* that can be configured. They are responsible for the content and its presentation. Modules are based on the *common base library*, which separates the personalization task from the session management. In script languages application control logic is usually comprised in one page, so the life cycle of that code is limited on that page or the active request. It is much better when the code is divided into external components. Then the life cycle of the component can be defined by the programer. These components can be used simultaneously by several requests and generated data is this way directly exchanged.

The components can, for example, exchange messages between user sessions using the standard synchronization methods. If the programming logic is split into several components, it can be used at the same time on the different places (*reusability*) or can be changed independently from the rest of the page (*updates*). Thus, using modules brings only improvements to the web software development.

4.1.2 Dynamic representation

The dynamic representation includes information selection, their further treatment in the program and the presentation of result. The problem that appears frequently in the application development is changing the output format. Usually application as an output directly produces just one presentation format, for example, HTML. In the situations, when it is necessary to transform application output to be WAP (*Wireless Application Protocol*) compliant, and that way make the portal accessible through mobile phone, usually the complete redesign is required. This means that information production and their representation are not separated.

4.1.2.1 INTERMEDIATE REPRESENTATION

The solution to the indicated problem is separation of information and presentation. The common feature of both HTML and WML outputs is information, used for production of the output format. This information has to be stored somewhere. Since, under normal conditions, this information holds some structure, employing the structured intermediate representation on the transformation level makes sense. What is the effect of using intermediate representation, depends the most on the level of interaction between the information producer (applications) and transformation level (consumer of the structured information). Information can be stored in the binary format, encoded, taken from the database or textual. When selecting the intermediate representation format, different factors should be considered, including the appropriate tools and working environment for manipulating this representation. Currently, for document representation on the web mark-up languages from XML/SGML class are used. Thus, it is obvious that the intermediate representation should also hold a XML alike i.e. tree structure. The only problem that stays unsolved is whether the format of the intermediate representation should be standardized, since there are situations where it makes sense to foreword produced information directly to the next level, in the binary format.

4.1.2.2 TRANSFORMATION

From the structured intermediate representation transformed output is generated (figure 4.4). The flexible transformation language enables simple information selection and modification. From the implementation point of view, the question of speed is very interesting.

4.1.2.3 SEPARATION OF CONCERNS

With the growth of web, the need for cooperation between people from different areas appears. Among them there are people from marketing, designers, artists, ergonoms, programmers and many others. Each of them, applying his/her specialized knowledge partially contributes to task completion. At the same time, each particular team member has to have the possibility to freely modify the parts, (s)he is responsible for. However, usually designers work on the *layout* comes to direct conflict with the work of the programmer. Every change of the code requires the change of the style, and so on. Therefore, the system that enables separation of the application into various interest spheres is necessary. This is done mainly by technologically imposing a reduced number of contracts and placing them in a hierarchical shape, suitable for replacing current high-structure web site management models. A model used in this portal prototype (also accepted by Cocoon [CO02]) is the *pyramid model of web contracts,* shown by figure 4.3, and is composed by four different working contexts:

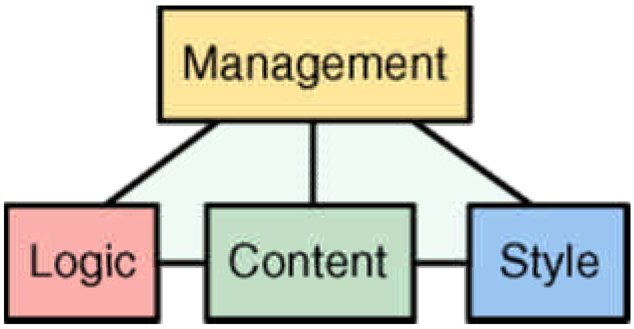


Figure 4.3 Pyramid of contracts

The working contexts are:

- *Management* – The people that decide what the site should contain, how it should behave and how it should appear.

- *Content* – The people responsible for writing, owning and managing the site content. This context may contain several sub-contexts - one for each language used to express page content.
- *Logic* – The people responsible for integration with dynamic content generation technologies and database systems.
- *Style* – The people responsible for information presentation, look & feel, site graphics and its maintenance.

The contracts that exist in the Cocoon model are: *management – content*, *management – logic*, *management – style*, *content - logic* and *content – style*. Note that there is no *logic - style* contract. The separation of concerns goal is completely consistent to the goal to partition the programming logic into modules.

4.2 Functions and Architecture

The portal receives requests from the multiple clients. It maps those requests onto various application calls. The portal applications generate responses in the intermediate representation format (refer to section 4.1.2.1); these intermediate representations are then transformed into appropriate output format and forwarded to the client. This means that the system, illustrated by the figure 4.4, must comprise of, minimum three levels: client, application and transformation.

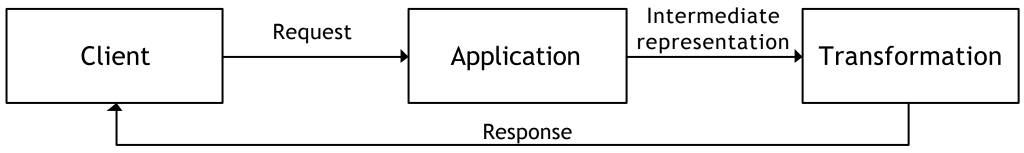


Figure 4.4 The basic portal components

4.2.1 Communication Level Architecture

The ***ETH World*** portal framework should be easy accessible through the existing clients, like web browser, for example. These clients are able to present HTML, XML and XHTML and use HTTP protocol. Due to large availability of these clients, communication protocol relies upon HTTP. This, at the same time, enables the existence of WAP applications, since WAP clients communicate only through ***WAP Gateway***, which, also, requests needed contents through HTTP. Utilization of the protocols, different from HTTP, is basically realizable through server-side adapter. The client is responsible for content representation and direct user interaction. All other aspects including security, transactions, load balancing, administering applications and so on are handled by the server. This kind of server is termed *Web Application Server*. Thus, the implemented portal is, in the same manner, referred to as *Portal Application Server*.

4.2.2 Portal Application Server

Server-side accepts the client `request` and returns the `response`. In the meantime all the applications, needed for the response generation, have to be called for execution; and their results transformed into required format. It is desirable that the applications are concentrated on the function of information production; technically, this is possible by establishing a central component. It controls applications, and at the same time offers them basic functionalities in the form of programming libraries. These functionalities are available through the Application Programming Interface (API) and the managing component (`Dispatcher-a`). API provides applications with the services within the following domains:

- Personalization (access to user and group parameters)
- Security (checking access rights)
- Calls to other applications (integration)
- Managing details of the HTTP response (redirection, cookies)

The coordination component (`Dispatcher`) is responsible for the management of overall processing, which includes the following functions:

- HTTP request dispatching
- Session management
- Initialization and application reload
- User identification and authentication
- Transformation of the intermediate representation into output format
- Error handling

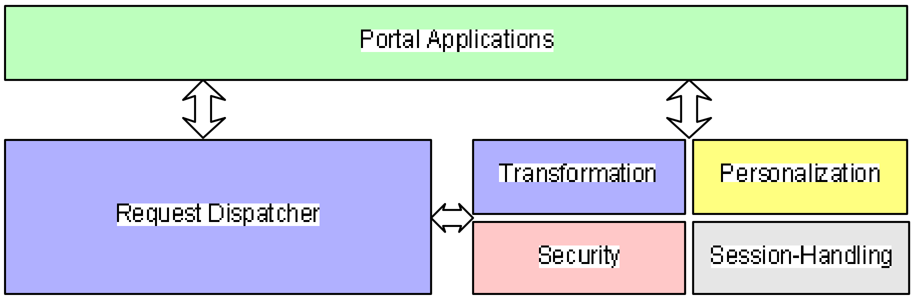


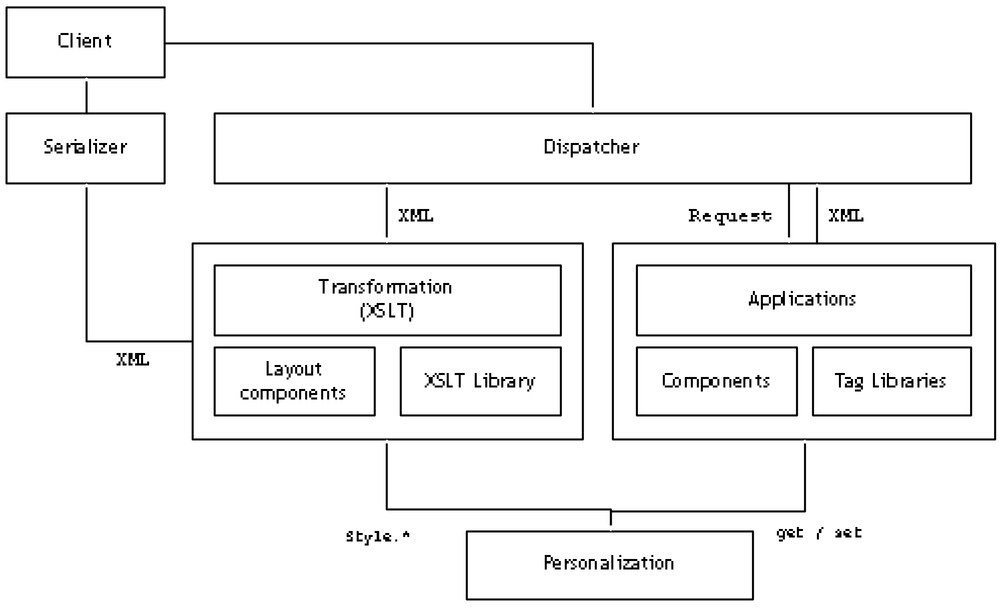
Figure 4.5 Building blocks of the portal application server

Block scheme of the portal application server is depicted by the figure 4.5, while detailed description of its architecture can be found in the following text.

4.2.2.1 DATAFLOW

Figure 4.6 illustrates the conceptual portal architecture (on the component level). In order to illustrate one of the possible scenarios the path of the individual client request is monitored:

- A client sends request through HTTP to the coordinating component (`Dispatcher`). The latter decides, which applications should be called on execution, and forwards request accordingly.
- Specific applications execute program logic. They can include or call on execution other components, tags from a tag library and so on. Through the portal personalization interface, user parameters for the specific application can be accessed (they can be loaded, set, modified, or deleted). The application output, actually an intermediate representation in XML format, is returned to the `Dispatcher`.
- `Dispatcher` forwards that output to the XSLT transformer. The main XSLT stylesheet can, using `import` and `include` directives, incorporate other stylesheets, which are stored in the libraries. All user parameters, from the `Style` parameter group become available as stylesheet parameters. Additional parameters can be forwarded, through the source document subtree, to the external transformer components, for more complex processing. In such a manner, for example, dynamic images can be viewed. Transformed representation is returned to the `Dispatcher`, in the XML format.
- All the applications should send their final outputs to the `Serializer`, which is responsible for the ultimate transformations (for example, HTML to XHTML) and results are sent directly to the client.



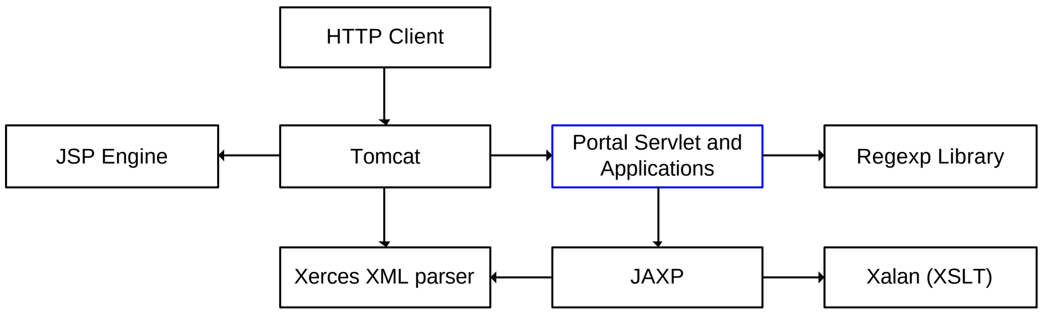


Figure 4.7 External Portal components

Figure 4.7 depicts a simplified scheme of the system components cooperation. Tomcat invokes the Portal servlet, which forwards `HTTP-Request` to the acquired application, in accordance with the mapping indicated in the portal configuration. The acquired application can include other applications. The application output (intermediate representation) is sent back to the portal servlet, and using the suitable stylesheet transformed with ***Xalan***. Transformed output is sent to the `Serializer` (this task is taken by the Xalan, too), and end result through ***Tomcat*** returned to the `HTTP-Client`. When the JSP application is referenced, first the *wrapper* application is called, which then forwards the request to the requested application. XML output of this application is automatically parsed by ***Xerces*** and in the ***DOM*** tree format sent back to the portal servlet. Further on, transformation and serialization are executed. In the following text there will be some word on the basic features and the way certain system components function.

4.2.3.1 SERVLETS AND TOMCAT SERVLET CONTAINER

I Servlets

Servlets [JSWP01] are protocol and platform-independent *server side* components from the, written in Java, which dynamically extend Java-enabled servers. Servlets interact with web clients via a *request/response* paradigm implemented by the *Servlet Container*. This request-response model is based on the behavior of the Hypertext Transfer Protocol (HTTP). Their initial function is to enable secure web-based access to data, which is presented using HTML web pages, interactively viewing or modifying that data using dynamic web page generation techniques.

Java Servlet API is a standard ***Java Extension API***, available as an add-on package. This package (Sun product) can be used to embed servlet support in other web servers, including ***Apache*** (and derived servers, such as ***Stronghold***), ***Netscape FastTrack*** and ***Enterprise*** servers and ***Microsoft***'s ***IIS***. When using *Servlet API* author is provided by all the advantages of Java language: not only the code will not have any memory leaks and suffer from hard-to-find pointer bugs, but it runs on platforms from many server vendors.

II Ways to Use Servlets

Although all servlets are written in Java, their clients may be written in any language. When servlets are used in middle tiers of distributed application systems, they can in turn be clients to other services, written in any language. For example, servlets can use JDBC™ to contact relation database. Communicating with other kinds of current or legacy systems may call to alternate software packages. A few of the many applications for servlets include:

- A simple servlet can process data, which was `POST`-ed over `HTTPS` using an `HTML FORM`, passing data, such as, purchase order (with credit card data).
- Since servlets handle multiple requests concurrently, the requests can be synchronized with each other to support collaborative applications such as *on-line* conferencing.
- One can define a community of active agents, which share the work among each other. Each agent should be implemented as a servlet, and agents would pass data to each other.
- One servlet could forward requests to other servers. This technique can balance load among several servers which mirror the same content. Or, it could be used to partition single logical service between several servers, routing requests according to task type or organizational boundaries.

III Servlets and Framework

Since servlets are Java objects, they have instance-specific data. This means that servlets are independent applications running within servers, without engaging additional classes (which are required by some alternative extension server APIs). During initialization servlets have access to some servlet-specific configuration data. This allows different instances of the same servlet class to be initialized with different data, and be managed as differently named servlets. Other means of interaction for servlet and their framework is utilization of the `ServletContext` object.

IV Servlet Container

The *Servlet Container*, in conjunction with a web server or application server, provides the network services over which requests and responses are set, decodes MIME based requests, and formats MIME based responses. A *Servlet Container* also contains and manages servlets through their lifecycle. *A Servlet Container* can either be built into a host web server or installed as an add-on component to a web Server via that server's native extension API. *Servlet Containers* can also be built into or possibly installed into web-enabled Application Servers. All servlet containers must support HTTP as a protocol for requests and responses, but may, also support other *request/response* based protocols such as HTTPS (HTTP over SSL).

A Servlet Container may place security restrictions on the framework that a servlet executes in. In a Java 2 Platform Standard Edition 1.2 (J2SE) or Java 2 Platform Enterprise Edition 1.3 (J2EE) environments these restrictions should be placed using the permission architecture defined by Java 2 Platform. For example, *high-end* application servers may limit certain action, such as the creation of a `Thread` object, to insure that other components of the *container* are not negatively impacted.

Tomcat 4 is an *open source* Java *Servlet/JSP Container*, deployed by Apache Software Foundation [APACHE]. This version implements Java Servlet API 2.3 and JSP 1.2 [JSP], and has numerous features that make it useful platform for web application and web service development. It consists of a large number of different components, which are connected and communicate according to the configuration file `conf/server.xml`. As already indicated, ***Tomcat*** takes over the lower foundation level of portal functionality and based on corresponding configuration data (element `Context` in file `server.xml` for `URI` specified by the `Host` element and file `web.xml` in the designated context) forwards requests from an external world to the Portal servlet component. Thus, ***Tomcat*** contains WebDAV-Filter [WDAV] that allows *distributed Authoring and Versioning* of web applications and documents.

V Servlet Lifecycle

Servlets are always dynamically loaded, although servers usually provide an administrative option to force loading and initializing particular servlets when server starts up. Servlets are loaded using normal Java class loading facilities, which means that they may be loaded from remote directories (for example, a trusted `https://department/servlets` directory) as easily as from local file system (figure 4.8). This increases flexibility in system architecture and easier distribution of services in a network. Loading is executed by the *Servlet Container's ClassLoader*. Servers also vary in how they know when to load servlets. When a request comes in, the server knows how to map it to a servlet, which may first need to be loaded. This mapping will usually be done in one of these common ways:

- Server administrators might specify that some of client requests always map to a particular servlet. For example, one which talks to a particular database.
- Server administrators might specify that part of the client request is the name of the servlet, as found in administered `/servlets` directory. At many sites, that directory would be shared between servers, which share the load of processing for the site's clients.
- Some servers may be able to automatically invoke servlets to filter the output of other servlets, based on their administrative configuration. For example, certain types of servlet output may trigger postprocessing by other servlets, perhaps to perform format conversions.

- Properly authorized clients might specify the servlet which is to be invoked, without administrative intervention.

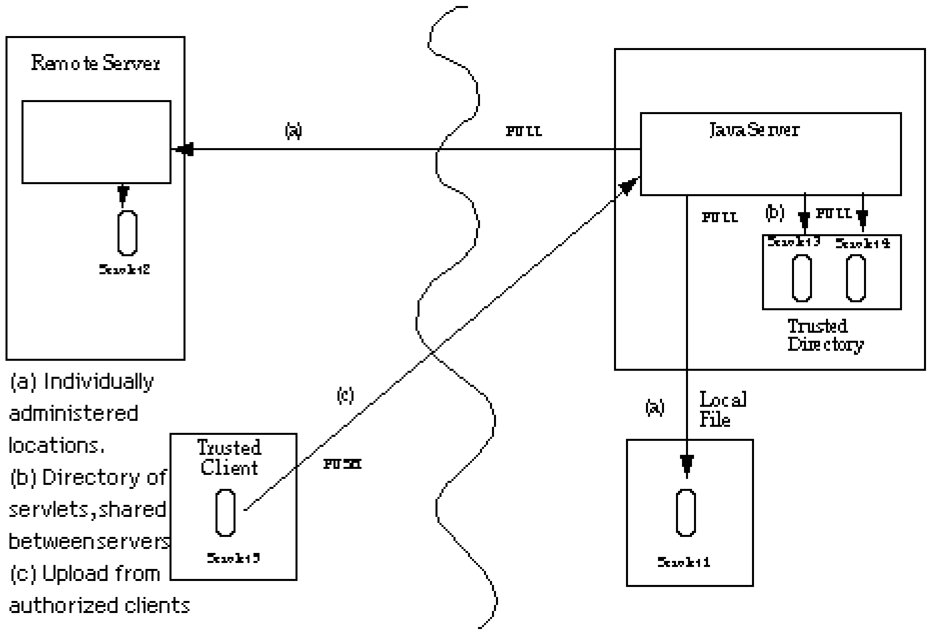


Figure 4.8 Possible sources of servlets

After being loaded, three methods are involved in the lifecycle of a servlet:

- Servlets are activated by the server through an *init* method call. Servlet authors may, if they want, provide their own implementation of this call, to perform potentially costly (usually I/O intensive) *setup* only once, rather than once per request. Examples of this kind of setting up are session initialization using other network services and providing access to permanently stored data (in a database or file).
- After initialization, servlets handle many requests. Each client request generates one call of method *service*. These requests may be concurrent; this allows servlet to coordinate activities among many clients. Class-static state may be used to share data between requests.
- Requests are processed until the servlet is explicitly shut down by the web server, by calling the *destroy* method. Then `Servlet` class is ready for *garbage collection*.

VI Security

Servlets have access to information about their clients. When used with secure protocols such as SSL, peer identities can be determined reliably. Servlets relying on HTTP, also have access to HTTP-specific authentication. Since servlets have the Java advantage memory access violations and strong typing violations are not possible, so that faulty servlets will not crash servers.

Unlike any other current server extension API, Java Servlets provide strong security policy support. This is because all Java environments provide a *SecurityManager*, which can control whether the actions, such as network or file access are to be permitted. The assumption is that all servlets are distrusted, and are not allowed to perform operations such as accessing network services or local files. However, servlets "built into" server, and servlets that have been digitally signed, as they were put into Java Archive (JAR) files, may be trusted and granted more permission by the *security manager*. A digital signature on executable code indicates that the organization, which signed the code "vouches for it" that it does not break security polices.

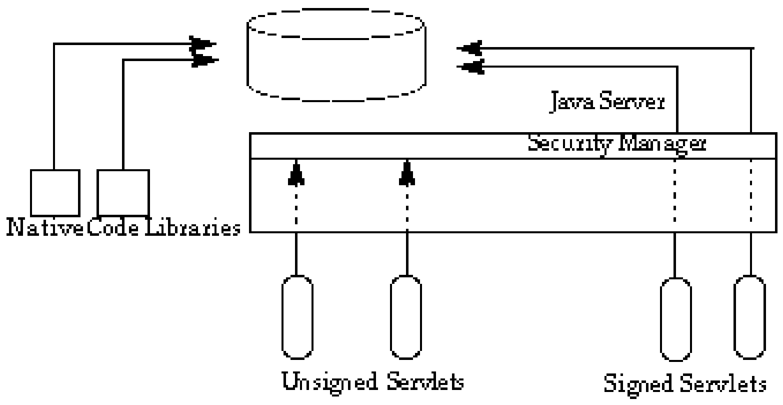


Figure 4.9 Comparing two approaches to server extensions

Figure 4.9 shows the comparison of two existing approaches to server extensions. One way is to let Java *SecurityManager* monitor at fine granularity servlet activities. In this case configuration of Java *Security Manager* has to be indicated in file `$CATALINA_HOME/conf/catalina.policy`, which is the part of ***Tomcat*** *Servlet Container* installation. This file completely overrides `java.policy` file that lies in JDK system directory. The second approach includes activities of *native code* extensions, which are never monitored. In both cases, a host operating system will usually be used to provide very coarse grained protection.

VII Performance

One of the biggest performance features of servlets is that they do not require creation of a new process for each request. In most environments, many servlets run in parallel within the same process as the server. When used in such environments with `HTTP`, servlets provide compelling performance advantages over both CGI and Fast-CGI approach. This is the consequence of the fact that servlets require *light-weight thread* context switches. Even Fast-CGI uses *heavy-weight* process context switching on each request, while regular CGI requires even heavier *start-up* and initialization code on each request. Figure 4.10 compares three server extension approaches:

- (a) The servlet approach is normally used to support embedding inside the server;

- (b) CGI uses a new child process per request;
- (c) Fast-CGI uses one child process for many requests.

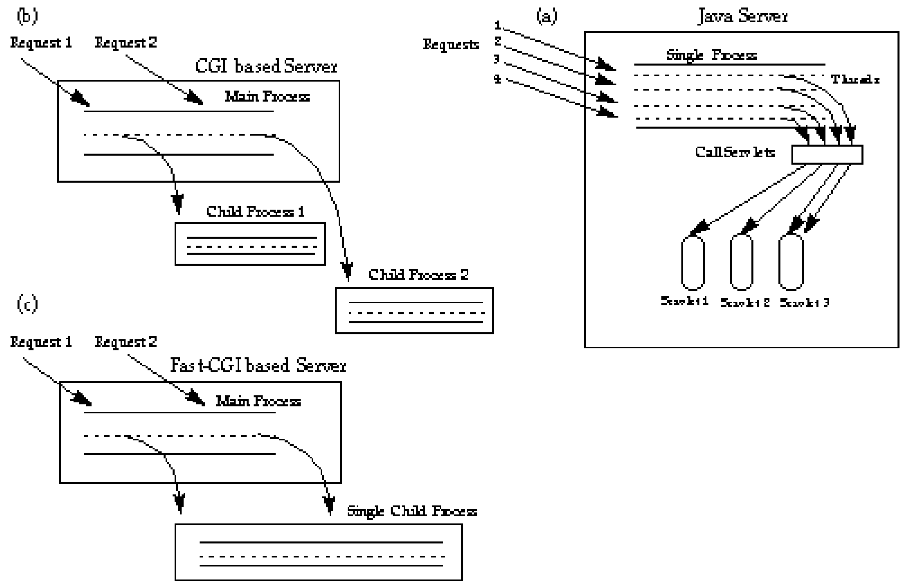


Figure 4.10 Comparison of three server extension approaches

Thus, for example, at the moment the server starts, *Connector* component of ***Tomcat*** *Servlet Container* creates specified number of *request processing threads*. One instance of this component listens the connection to specific TCP port on the server. One or more *Connectors* may be configured, as a part of one *Service* and each of them accepted request forwards to the assigned *Engine* element, which then processes that request and creates response. The created response is over the same *Connector* returned to the client. Each accepted request requires one thread for execution. If the server receives simultaneously more requests than it can handle using the available threads, the additional ones can be created (until maximum number specified by `maxProcessors` attribute of the `<Connector/>` element in `conf/server.xml`). Additional number (`acceptCount`) of incoming simultaneous requests is accepted to wait for the server socket to get released, while all the rest gets "connection refused" error message.

Since in most environments already initialized servlets can handle many client requests, the costs for initialization are spread over many methods. All the client requests to that service have the opportunity to share the data and communications resources, benefiting more strongly from system caches. With many implementations of the Java Virtual Machine (JVM), Java Servlet applications automatically take advantage of additional processors. This helps provide better throughput and response time to the clients. Because 100% pure Java programs do not care what operating system they use, author has the freedom to choose whatever system vendor best addresses his/her application requirements.

4.2.3.2 XERCES-J 1.3

I DOM

For access and processing of XML documents in applications corresponding interface is necessary. As a standard W3C recommended ***Document Object Model*** (***DOM***). Class `DocumentBuilder` parses original document and builds in memory internal presentation of the whole document as a tree. ***DOM*** defines API for accessing that tree. Among other things, in the application it is possible, to select and modify parts of the tree, as well as specific elements, their deletion from the tree, or adding new elements. Thus, it is frequently used in the applications that access configuration files.

Example 4.1 DOMParser

```
...
public static void main(String args[]){
   DocumentBuilderFactory factory =
      DocumentBuilderFactory.newInstance();
   DocumentBuilder domBuilder = factory.newDocumentBuilder();
   configDoc = domBuilder.parse( "configuration.xml" );
   //in the document retrieves a dbconnection node
   Node dbconn =
      configDoc.getElementsByTagName("dbconnection").item(0);
   // finds a list of attributes
   NamedNodeMap attributes = dbconn.getAttributes();
   // and for each attribute in the list
   while (attributes.getLength()> 0){
      Node currentAttr= attributes.item(attributes.getLength()-1);
      String name = currentAttr.getNodeName();
      // generates corresponding element that is
      Element newEl=configDoc.createElement(name);
         newEl.appendChild(
         configDoc.createTextNode(currentAttr.getNodeValue()));
      // as a child element added to dbconnection node
      dbconn.appendChild(newEl);
      // anhd then delets that attribute from the attribute list
      attributes.removeNamedItem(name);
   }
   // Serializes DOM
   OutputFormat format = new OutputFormat(configDoc);
   // writes modified document into original file
   FileWriter stringOut=new FileWriter("configuration.xml");
   XMLSerializer serial = new XMLSerializer(stringOut,format);
   // as DOM Serializer
   serial.asDOMSerializer();
   serial.serialize(configDoc.getDocumentElement());
}
```

The example 4.1 sows simple application that uses ***DOM*** parser. It parses and modifies file `configuration.xml`. In the application node named `<dbconnection/>` is accessed and all its attributes transformed into elements, and then the original tree is replaced by the modified one in the original document.

In the example 4.2 the element `<dbconnection/>` is shown, before and after the modification is executed. The same effect can be achieved when using ***XSLT*** and ***Xalan*** as a processor.

Example 4.2 configuration.xml before the modification

```
...
<!– database connection-->
<dbconnection application="dispatcher"
driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
init="jdbc:microsoft:sqlserver://gorillaz:1433;
Database=TryCourseScheduler;SelectMethod=cursor"
user="sunpress" password="sunpress" connectionCount="2" />
...
```

configuration.xml after the modification

```
...
<!– database connection -->
<dbconnection>
    <user>sunpress</user>
    <password>sunpress</password>
    <init>jdbc:microsoft:sqlserver://gorillaz:1433;
    Database=TryCourseScheduler;SelectMethod=cursor</init>
    <driver>com.microsoft.jdbc.sqlserver.SQLServerDriver</driver>
    <connectionCount>2</connectionCount>
    <application>dispatcher</application>
</dbconnection>
...
```

II SAX

Simple API for XML [SAX] functions differently: instead of building the document tree, during the parsing process it generates the series of events. Application registers `EventHandler` for these events and receives from it information on document and its structure, based on which it builds corresponding internal representation. Thus, for example, parser generates events when encountering the beginning or the end of the document, start and end tag of the element, … Therefore, ***SAX*** is a good choice in the situations when application needs just a part of the original XML document; as a matter of fact, it can be used as a filter. Example 4.1, implemented using ***SAX*** parser, is much more complex and requires much more space, so it is given in the appendix A.

III SAX vs. DOM

Because of the *event-driven* nature of the program `EventHandler` usually has to handle additional lists and counters that store information on the state of document. Thus, it is necessary to be careful when more threads use the same `EventHandler` simultaneously. Working with ***DOM*** is much easier. It is convenient when application modifies source document, i.e. when application does not require internal data representation but transforms already existing DOM tree. Problem appears when the source document is too big, and then DOM tree occupies a large part of the memory and slows down the application.

Xerces is the most common XML Parser, also developed and made available by the Apache Software Foundation [XERCES]. In this package both, DOM Level 2 [DOM] and SAX Version 2.0 [SAX], are implemented since there are situations when each of them shows significant advantages. The ETHWorld portal uses just

DOM functionality in order to access configuration data and for integration of the XML output of one application in the other.

There is another API for accessing XML documents; it is not implemented as a part of ***Xerces*** package, but as a separate ***JDOM*** package (refer to section 4.4.).

Example 4.3a SimpleTransformer

```
public static void main(String args[]){
   DocumentBuilderFactory factory =
      DocumentBuilderFactory.newInstance();
   DocumentBuilder domBuilder = factory.newDocumentBuilder();
   configDoc = domBuilder.parse("configuration.xml");
   //in the document retrieves the node named dbconnection
   Node dbconn =
      configDoc.getElementsByTagName("dbconnection").item(0);
   TransformerFactory tf = TransformerFactory.newInstance();
   // Creates transformer for indicated stylesheet
   Transformer modify =
      tf.newTransformer(new StreamSource("modify.xsl"));
   // transforms and writes out into source file
   modify.transform(new DOMSource(configDoc),
      new StreamResult(new FileWriter("configuration.xml"));
}
```

b) Modify.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/..." version="1.0">
   <xsl:template match="/">
      <xsl:apply-templates select="*"/>
   </xsl:template>
   <xsl:template match="dbconnection">
      <xsl:element name="{name(.)}">
         <xsl:apply-templates select="@*"/>
      </xsl:element>
   </xsl:template>
   <xsl:template match="*">
      <xsl:element name="{name(.)}">
         <xsl:for-each select="@*">
            <xsl:attribute name="{name(.)}">
               <xsl:value-of select="."/>
            </xsl:attribute>
         </xsl:for-each>
         <xsl:apply-templates/>
      </xsl:element>
   </xsl:template>
   <xsl:template match="@*">
      <xsl:element name="{name(.)}">
         <xsl:apply-templates/>
      </xsl:element>
   </xsl:template>
   <xsl:template match="text()">
      <xsl:value-of select="."/>
   </xsl:template>
</xsl:stylesheet>
```

4.2.3.3 XALAN-J 2.0

Xalan is a stable and extendable XSLT processor implementation [XALAN]. ***Xalan*** is one of the first XSLT processors, compatible with ***JAXP*** transforming interface. By using the Xalan XSLT-processor in the portal intermediate representation is transformed and serialized.

Example 4.3a shows a part of the `main` method (without `try/catch` brackets and declarations) of a Java application that calls on ***Xalan*** to transform source XML document. The transformation process gives the same result as the code in example 4.1, though the way of getting the result is different. When using Xalan XSLT stylesheet for transforming XML document is needed (shown in the example 4.3b).

4.2.3.4 JAXP 1.1

Java API for XML processing [JAXP] is an abstract interface, which guarantees compatibility with different XML parsers and XML-transformers. API is consisted of the parsing part and the transformation part. JAXP extends existing DOM API with new functionalities: provides method for reading DOM `Document` object from XML tree, methods for controlling parser behavior (validation and error handling), and provides pluggable DOM parser implementation. The transformation part contains specialized implementations of interfaces `Source` and `Result` in `javax.xml.transform.*` packages.

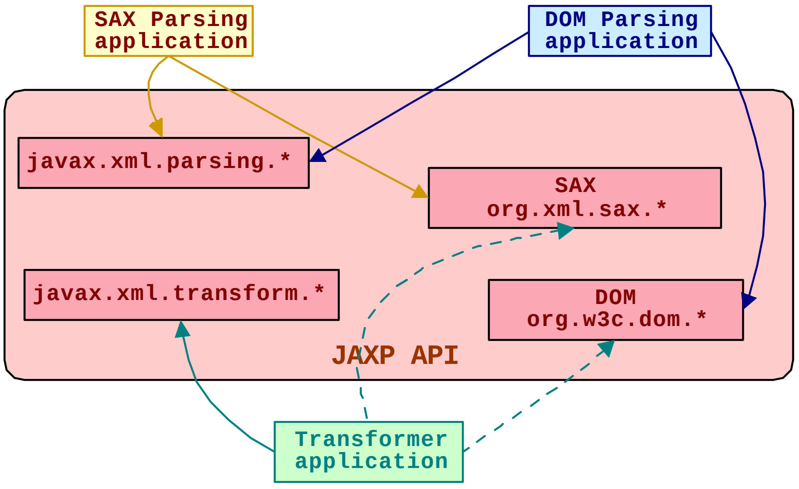


Figure 4.11 JAXP API and its utilization

One JAXP usage is shown in the example 4.3, where method `newInstance()` from `abstract` class `javax.xml.transform.TransformerFactory`, is referenced as a part of JAXP; while at the runtime real implementation for that class is dynamically loaded, according to data specified in configuration file (for example, it may be `org.apache.xalan.processor.TransformerFactoryImpl`, implemented in ***Xalan***). JAXP contains four modules that are activated, depending on the task (figure 4.11).

By using ***JAXP*** interface, it is possible to simultaneously embed different XML parsers and transformers, compliant to ***JAXP*** API, into portal.

4.2.3.5 JSP 1.2

Java Server Pages Specification [JSP] enables creation of dynamic pages (mixture of tags and Java code, refer to section 4.4.1). They are, on request, parsed and compiled into Java servlet, which in a row receives that request from JSP processor. One of the basic JSP features is separation of code and layout, by using Java Beans [EJB] and Java tag libraries. In the portal JSP is used for application development (JSP generates XML).

4.2.3.6 Jakarta Regexp

For correct URI mapping onto appropriate application, `Dispatcher` (package `portal.framework`) uses `RegularExpression` Java library [REG]. Thus, it is not necessary to use regular expressions when writing applications, which offers the possibility to use more complex mappings later on.

I org.apache.regexp.RE

RE is efficient lightweight class that evaluates and maps regular expressions. Regular expressions are pattern descriptions, which enable sophisticated string mapping. In addition to being able to match a string against a pattern, it is also possible to extract parts of the match. This is especially useful in text parsing.

To compile a regular expression (RE), simple construction of an RE matcher object from the string specification of the pattern is made:

```
RE r = new RE("a*b");
```

Once this is done, either of `RE.match()` methods should be called to perform matching on a String, for example: `boolean matched = r.match("aaaab")` sets the `boolean` variable `matched` on `true`, since string "aaaab" matches the pattern "a*b" … `RE` runs programs compiled by the `RECompiler` class. Class `RE`, for reasons of efficiency, does not include the actual regular expression compiler; thus, when one or more regular expressions need to be pre-compiled, the 'recompile' class can be invoked from the command line to produce compiled output like this:

```
// Pre-compiled regular expression "a*b"
char[] re1Instructions = {
    0x007c, 0x0000, 0x001a, 0x007c, 0x0000, 0x000d, 0x0041,
    0x0001, 0x0004, 0x0061, 0x007c, 0x0000, 0x0003, 0x0047,
    0x0000, 0xfff6, 0x007c, 0x0000, 0x0003, 0x004e, 0x0000,
    0x0003, 0x0041, 0x0001, 0x0004, 0x0062, 0x0045, 0x0000,
    0x0000,
};
REProgram re1 = new REProgram(re1Instructions);
```

Then, regular expression matcher object (`RE`) from the pre-compiled expression `re1` can be constructed and thus avoid the overhead of compiling the expression at runtime. If more dynamic regular expressions are required, a single

`RECompiler` object can be constructed, and re-used to compile each expression. Similarly, it is possible to change `REProgram`, run by a given `matcher` object at any time.

In the portal this package is used when calling applications, it maps URIs on to application calls. Patterns (application names) are stored in the `confguration.xml` file (refer to section 4.3).

4.2.3.7 PACKAGES

Portal Application Server contains over 30 classes and interfaces, organized in 7 packages. Class `Dispatcher` from the package `framework`, which implements servlet functionality, overtakes handling the incoming requests, as well as global portal management. The rest of the classes and interfaces, are organized in following 7 packages:

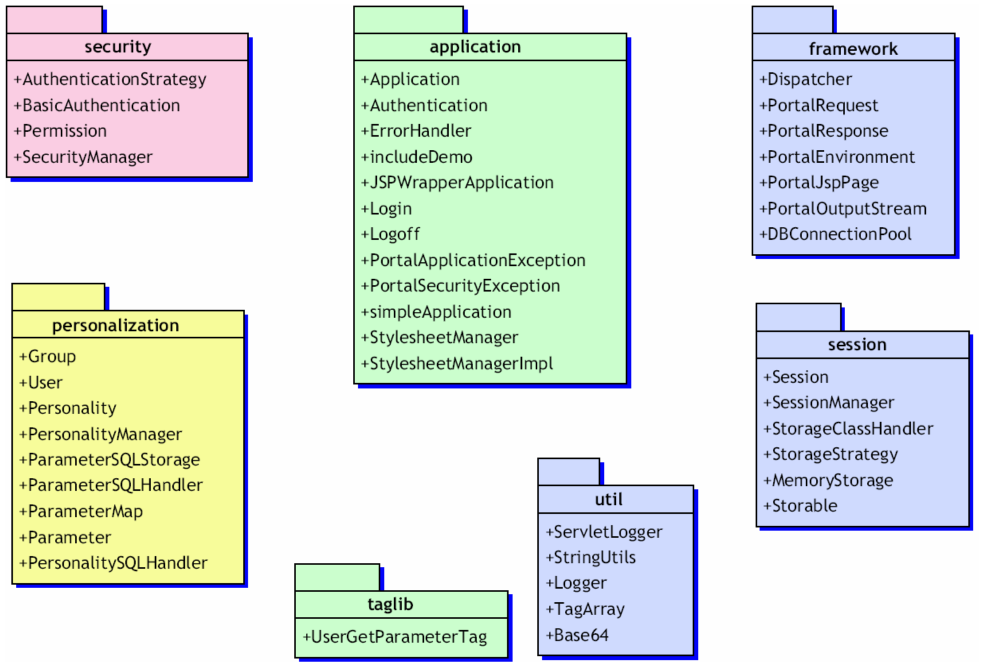


Figure 4.12 The portal package diagram

- *Framework:* Supervision of the portal operation. Communication with *Servlet Container* and database. Provision of wrapper classes for `Request` and `Response` objects.
- *Session:* Generation and managing `Session` objects.
- *Application:* Base classes in Portal for application execution. Maintains stylesheets, default applications (`Login, Authentication, Logout, Logoff, ErrorHandler, JSPWrapperApplication`). `Exeption` classes for error handling.
- *Security:* Checking document security privileges and authentication.
- *Personalization:* User and user group administration. Implementation of hierarchical user parameter storage.

- *Util:* Utility classes for login, string processing, Base64 encoding tag-array conversion.
- *Taglib:* Tag libraries to be used by the Portal JSP applications.

In the following text several chosen classes are described. From the description it is possible to get the impression where which functionality was implemented. Actual implementation details can be identified in the code.

I *framework.Dispatcher*

Class `Dispatcher`, implemented as a servlet, represents the main portal class; thus, it is addressable for clients from the outside world by using `HTTP`-protocol. As already indicated, it overtakes total control over the portal and applications. The main tasks of this class are:

- *Initialization.* By referencing `init` method, *Servlet Container* enables the portal to initialize its own variables. First the configuration is loaded and parsed from the `configuration.xml` file. Then, database pooling component is created and configured, and some of the instances of `Transformer` or `Serializer Factory` classes initialized. Class `PortalContext`, which is created later, contains functionalities that can be used by all applications and the framework itself (for example, XML Parser or database-pool instance). In the steps to follow *Java Classloader* loads application classes and creates one instance of each. One instance of a `StylesheetManager` is placed on the site, to administer stylesheets required by applications. Further on, using the `RegularExpression` class URI `<mapping/>` is compiled, to enable execution of the corresponding applications. The `Dispatcher` servlet contains an instance of each of the remainder system applications (`SessionManager, SecurityManager, PersonalityManager`).
- *Request mapping and routing:* When `Dispatcher` receives the request from the *Servlet Container*, it locates, according to the element `<mapping/>` in the file `configuration.xml`, an appropriate application and forwards it that request. The application itself can, in a row call `Dispatcher` in order to integrate other applications.
- *User identification and authentication:* Prior to the request forwarding to the appropriate application, it is checked whether identification or authentication are required. If they are required, and current user does not have the requested access rights, call to needed application is replaced by the class `Login` or `Authentication` call. They are also implemented as applications.
- *Transformation and Serialization:* An application output has to be transformed by using the suitable stylesheets. `Dispatcher` determines which stylesheet to use according to the content type of the `PortalResponse` object and calls on the appropriate `Transformer`. Before the output is sent to the client `Serializer` is addressed to, from internally used DOM-tree create

text and write it to stream. Later on, the conversion is executed compliant to the specific rules (for example, character encoding, XHTML from HTML).

II session.Session Manager

Administration of user sessions is taken over by the singleton `SessionManager` class. It offers functions like creation, loading, storing and deleting `Session` objects and discovering `SessionIDs`. `StorageStrategy` object determines the way persistency is implemented, and in the current version only `MemoryStorage` is offered. This object stores all objects into one instance of the `HashMap` class in memory.

III session.Session

One instance of the `Session` class contains all the information on one user session. Attributes can be read, written and it is possible to form the user hierarchy. In order to make the later integration of the portal and *Servlet Container* session management easier, class `Session` is derived from the `HttpSession` class.

IV personalization.Personality

`Personality` is either one `User` or the user `Group`. Each user has his/her own unique ID and belongs to only one group. The difference between two indicated subclasses is that they enable different administration of the group hierarchy. Function `getAllIds()` returns the list of all personalities, no matter whether they are users or groups.

V personalization.PersonalityManager

Singleton class `PersonalityManager` takes over the creation of `Personality` objects, like `User` or `Group`, for example. Validity of the objects is checked in the portal database and the appropriate groups are set.

VI framework.PortalRequest

With each reference to the application contained in the portal one class of the `PortalRequest` object is instantiated. This class wraps `HttpServletRequest`. When using this class application is enabled access to the parameters in the `HTTP` request or `HTTP` header. This wrapper class is needed, because only *Servlet Container* has the privilege to create and/or change `HttpServletRequest`. In the portal, however, it is necessary to have the ability to create new requests, for example, when recursively embedding applications or when the request is, using the wrapper application returned to the *container* (refer to the description of the `JSPWrapperApplication` class). Class `PortalRequest` inherits class `HttpServletRequestWrapper`.

VII framework.PortalResponse

Just like the `PortalRequest`, class `PortalResponse` is a *wrapper* class too; but it wraps class `HttpServletResponse`. When working with servlets redirection of the `OutputStream`, is usually not possible. However, the portal needs this functionality to enable forwarding the application output to XML parser. This is possible when class `HttpServletResponseWrapper` is used.

4.2.3.8 Session Manager and Personalization

Session management is one of the basic portal functions. The majority of *Servlet Containers* provides already defined session functionality. If the author uses this functionality she/he is restricted to interfaces defined by Java Servlet specification, but it can use suitable extensions of the actual *Servlet Container*.

Actual interface does not provide the possibility to choose freely the persistence mechanism (memory, database), which could represent the disadvantage when scaling up the system. Additionally, method for `SessionID` transfer (`URL` or `Cookie`) cannot be dynamically selected and therefore personalized. These are the reasons why for the given portal implementation new `SessionManager` (`portal.session.Session`) is developed. When using *Servlet/JSP Container* the portal has to use its own *SessionHandling*, due to lack of integration facility. This problem is possible to solve only by implementing new session classes for the actual *container.*

4.2.3.9 Security

Security is in this portal implementation simply realized on purpose. Passwords are stored in plain text format (instead hash function utilization) and there is only one authentication option, by using user name and password. The possibility of securing the *Servlet Container* is not taken into consideration. It would be reasonable to offer new mechanisms for authentication, like ones using `HTTP` *basic authentication* or analyze `SSL` *client* permissions.

4.2.4 Example dataflow

Being the main servlet, `Dispatcher` is responsible for correct calls of applications, that the current user holds appropriate privileges and for output generation. Since the servlet initialization process was clarified in the previous section, here the way specific requests are handled will be considered in detail. In the further text, the processes, occurring after the *Servlet Container* executes `Dispatcher` servlet `doGet` method call, are explained:

- *Connection to the database.* At the moment the application is started it demands the connection to the database from the *database pool*, and opens a new transaction. The main purpose of this database connection is consistent read and write of user parameters. This means that each request is executed alone in the database. Since the applications are sequentially processed, this is also applicable to each specific application.

- *Session.* The attempt is made to derive a valid `SessionID` from the available `HTTP` parameters. If the attempt is successful, `SessionManager` activates the session and `Dispatcher` is forwarded the instance of the `Session` class. `SessionManager` takes into consideration all `HTTP GET` and `POST` parameters and `Cookie` values with name *portalsession*. If the session does not exist, or is not valid any more a new session is created by calling the `newSession()` method.
- *User.* In this step user is identified and a new `User` object created, or the existing one retrieved. If the user is already connected with one session, the retrieval is finished. Any other way `PersonalityManager` is required to identify the user from the `Request portalUser` parameter. If this action ends unsuccessfully, which resembles to the level 0 on figure 4.1, object `User` is assigned *null* value.
- *Mapping.* Configured application mapping is used (refer to section 4.3), to identify appropriate application from the request URI.
- *Saved Request.* If the request redirection is needed, due to lack of user identification or in order to perform internal authentication, by using applications `Login` or `Authentication`, it is necessary to buffer original request. Hence, the actual user request is stopped and a new one cannot be created. If the outcome of `Login` application was successful, URL of the saved request is used to perform suitable HTTP redirection.
- *Permissions.* `SequrityManager` checks user access rights for all available resources.
- *Application call.* After the private method `processApplication()` is referenced, corresponding application starts execution. By calling method `handleRequest()` appropriate `PortalRequest` and `PortalResponse` objects will be created. Application can, by calling `includeApplication` method (class `PortalContext`), integrate transformed, but not yet serialized output of other applications.
- *Transformation.* Intermediate representation in XML format, generated by the application, is sent to `Dispatcher` as a part of DOM tree (`DocumentFragment`). First a new `Transformer,` compliant to JAXP 1.1, is generated for the defined stylesheet. Then all the parameters from group `Style` are set into this `Transformer` instance. Finally, DOM - DOM transformation is started.
- *Commit.* User parameters utilized in `User` object are, if changed, written back to database. After the execution of database `Commit` statement, the connection is released.
- *Serialization.* If there is no `HTTP` redirection (`HTTP` header field `Location`) then prepared output can be sent back to the client. `Serializer` is, actually, one usual transformer, which writes its output directly to `OutputStream`.

What type of output format is expected (XML, XHTML, HTML etc.) is already specified in the `OutputProperties`.

- *End.* `SessionManager` should now store the active session. For that, class `session.StorageStrategy` is used.

4.2.5 Database

4.2.5.1 INTERFACE

Pooling. Database connections are expensive resource. Primarily, because opening new database connection can take a while, and also because number of available database connections is limited (memory, performance, permissions). Therefore portal uses, so called database connection pool (*DBConnectionPool*). Once opened and initialized, connections can be taken from and returned to the pool after they are used.

Transaction. Since ***Java Database Connectivity*** (JDBC) allows using only one connection for maximum one transaction, each `Request` needs at least one connection, which reads and writes personal parameters. Since `Login` process (checking user name and password) is implemented as an application, it requires its own database connection. This application gets the connection from the pool, too. This means that existence of, at least, two open database connections is necessary for the correct portal functioning.

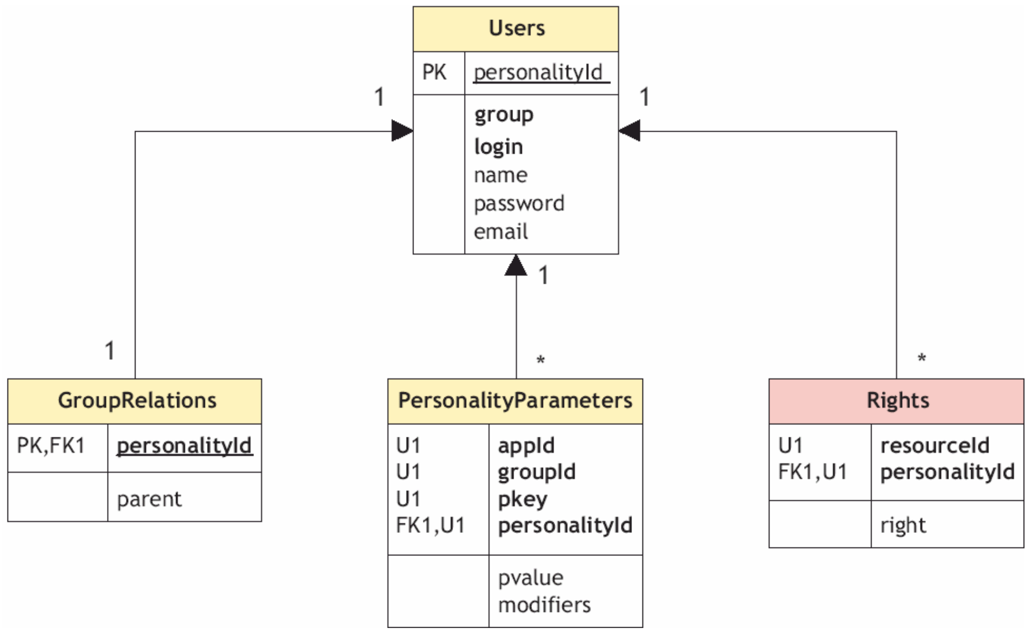


Figure 4.13 Relational database diagram

Threads. Database pool can be used simultaneously (servlets are *multithreaded* applications). Thus, methods `getConnection()` and `releaseConnection()` have to be `synchronized`. Threads, mutually, communicate by using `wait/notify` mechanism. One thread has *timeout* of 10 s, when it waits for the database connection. After this period the connection management process is blocked and the user receives an error message.

4.2.5.2 SETUP

DB scheme. Relational database is used as a permanent storage for user data, including personalization parameters, metadata on user and user groups, security features. Figure 4.12 illustrates relational database diagram. For the simplicity purposes this database was developed in ***Microsoft Access***, which does not support transactions (and afterwards exported to ***MS SQL Server 2000***). Any database, that supports JDBC, can be configured based on this example.

4.3 Portal Configuration and Portal Application Development

Portal files are situated in the various directories. Compiled Java classes are stored in `WEB-INF/classes` or `WEB-INF/lib` (as JAR files). Besides that `/lib` directory must contain `jakarta-regexp-1.2.jar`. Then, configuration of Portal servlet directory in ***Tomcat*** must be performed. This is done by insertion of the following lines in `<Engine/>` element of the `$CATALINAHOME/conf/server.xml` file:

```
<!—Tomcat Portal Root Context -->
<Context path=”” docBase=”E:/myDevelopment/myPortal”
debug=”1” reloadable=”true”>
```

From the security reasons it is recommendable, that from the file `web.xml` element `<Mapping/>`, which maps end *jsp* to servlet *jsp*, is deleted. `JSPWrapper-Application` calls servlets by using only their names. After the installation of the Portal servlet is performed ***Tomcat*** can be normally started. File `configuration.xml` is used as a central portal configuration file. It is stored in the servlet *document Root*. Beside all the other information, it contains definitions of all stylesheets and all applications.

I Base

Basic regulations related to the database and application *root* directory (without them set, system would not be able to function) are clarified by the tables 4.1 and 4.2. The corresponding examples illustrate the way they are defined:

```
<dbconnection application="dispatcher"
driver="sun.jdbc.odbc.JdbcOdbcDriver"
init="jdbc:odbc;User=portal;PW=portal"
connectionCount="2" />
```

Table 4.1. Portal and database connection

`Applications`	The list of applications, which can access database using dB-pool (*had not been considered in prototype*)

`Driver`	name of the Java class for the corresponding JDBC driver
`Init`	initialization database (here: ODBC for data source, user name and password)
`connection-Count`	Number of available connections in the pool i.e. simultaneously open connections.

```
<docroot path="E:/development/da/portal/applications"
relative="/applications" />
```

Table 4.2 Application Root Directory

`path`	absolute path to the directory containing applications and stylesheets
`relative`	path relative to Servlet-Root

II Transformers, Serializers

Definition of all JAXP transformers and serializers used is shown by the tables 4.3 and 4.4.

```
<transformer name="xslt"
class="org.apache.xalan.processor.TransformerFactoryImpl"/>
```

Table 4.3 Definition of one JAXP transformer

`Name`	unique identifier, assigned to a stylesheet during the definition
`Class`	complete name of the class that implements Transformer Factory.

```
<serializer type="text/html"
properties="method=html,ident-amount=2"
class="org.apache.xalan.processor.TransformerFactoryImpl"/>
```

Table 4.4 Definition of one JAXP serializer

`type`	MIME output type (unique) to be used by this Serializer
`properties`	Output characteristics, which re set at runtime.
`class`	complete name of the class implementing Factory (similar to transformer)

III Applications

Contains definition of all the available applications. They can be: built-in and add-on applications.

```
<!—login application -->
   <application name="login" class="portal.application.Login"
   path="login" authRequired="no" loginRequired="no">
</application >
```

Table 4.5 Description of one application

`Name`	unique identifier used for mapping
`Class`	complete name of the application class
`Path`	relative path to application directory, containing stylesheets and other data required by applications
`authRequired =[yes/no]`	user authentication (level 2) required or not
`loginRequired =[yes/no]`	user identification (level 1) required or not

IV Styles

Each application can be assigned one or more stylesheets. Definition of one stylesheet has the following format:

```
<style name="loginStyle"
   transformer="xslt" target="text/html" media="screen"
   file="login.xsl" default="yes"/>
```

Table 4.6 Connection to one stylesheet

`name`	unique identifier that can be used by application
`transformer`	ID of the Transformer-Factory (see above)
`target`	MIME output type and ID of the Serializer-Factory
`media`	Output medium for which the stylesheet is created (*had not been considered in prototype*)
`file`	file, that contains stylesheet indicated
`default=[yes/no]`	for each output format the default stylesheet can be defined

V Mapping

Defines mapping from an URI to an application:

```
<mapping pattern="/login*" path="/login" application="login"/>
```

Table 4.7 One mapping

`pattern`	Regular Expression used for matching the URI
`Path`	absolute mapping path, based in servlet root
`application`	ID of the application called

4.3.1 Applications as Java classes

All the applications, executed on the prototype, are basically Java classes, derived from `portal.application.Application`. This abstract superclass, defines interfaces for application calls and contains configuration information and application stylesheets. Class `Dispatcher` calls the method of this class:

```
public DocumentFragment
   handleRequest(PortalRequest req, PortalResponse resp)
             throws PortalApplicationException
```

this will be forwarded to the appropriate application, where `PortalRequest` and `PortalResponse` are wrapper classes for `HttpServletRequest` and `HttpServletResponse`, available to `Dispatcher` servlet. The output of the application is `DocumentFragment`, i.e. a part of a XML tree, compliant to DOM specification. `Dispatcher` receives `DocumentFragment`, transforms it, by using one of the, output type conforming, stylesheets and returns it to the client.

Development of an application, as Java class, is illustrated simple example `HelloWorld` output. Application output is in HTML format, and the procedure is following:

- Write an application that returns 'Hello World' as `DocumentFragment`. Text becomes the content of the structural component (`<hellotext/>`).
- Write a XSLT stylesheet that selects structural component and generates complete HTML output.
- Configure the application in the portal configuration file (`configuration.xml`) and restart ***Tomcat***.

Example 4.4 Hello World application (part)

```
public DocumentFragment handleRequest (
PortalRequest req, PortalResponse resp){
    Document doc = resp.getDocument();
    DocumentFragment docPart=doc.createDocumentFragment();
    Element docRoot = doc.createElement("hellotext");
    Text t=doc.creatTextNode("Hello World");
    docRoot.appendChild(t);
    docPart.appendChild(docRoot);
}
```

As it is obvious from the code example 4.4, in the first step global document for this requests `Response` is acquired. It is used to generate suitable `DocumentFragment`, `Element` and `Text`. Then the `Text` is as a *child* element, assigned to `<helloText/>` structural component, and this one to the `DocumentFragment`.

- *Stylesheet.* Appropriate XSLT stylesheet is shown by code example 4.5. As an illustration here two equivalent templates are defined (instead just one for `<hellotext/>` element). The first template generates HTML frame for the document, while the second one shows the content of `<hellotext/>` element (`value-of`). Stylesheet will be stored in the `/hello` directory in `Portal Application Root`, under *HelloWorld.xsl*.
- *Configuration.* The only thing left to do is application configuration in the portal. In the subtrees `<applications/>` and `<mappings/>` the following node has to be inserted:

```
<application name="HelloWorld"
  class="portal.demo.HelloWorld"
  path="/hello"
  authRequired="no" loginRequired="no">
  <style name="helloStyle" transformer="xslt"
     target="text/html" media="screen"
     file=" HelloWorld.xsl" default="yes"/>
</application>
<mapping pattern="/hello*" path="/hello" application="HelloWorld"/>
```

This provides that both, application and stylesheet are acquired when the portal is started. Prior to that application must be compiled and its class is stored in `WEB-INF/classes/demo`. By entering an URI `http://my.portal/hello` in browser address bar the execution of `HelloWorld` example is started.

Source code 4.5 Hello World-XSLT Stylesheet

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns=http://www.w3.org/... version="1.0">
   <xsl:template match="/">
      <html>
         <head><title>Hello Sample</title></head>
         <body>
            <xsl:apply-templates select="hellotext"/>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="hellotext">
      <h1><xsl:value-of select="." /></h1>
   </xsl:template>
</xsl: stylesheet >
```

4.3.1.1 WRAPPER APPLICATIONS

Generic wrappers. Using *wrapper* applications external information sources can be integrated into portal. These applications, usually, contain URI of the information source as an `HTTP Parameter` or as an additional path. They, alone take care on correction of mapping, loading referenced information sources and their parsing into appropriate DOM tree. This generic *wrapper* can be used several times and with different applications. As an example, an `XMLWrapperApplication` will be explained. It takes one XML document from the directory `PortalDocuments/RelativePath`. Thus, one of the possible definitions of this application is:

```
<!—hello world demo application -->
<application name="xmlfile"
class="portal.demo.XMLWrapperApplication" path="/"
authRequired="no" loginRequired="no">
</application >
<mapping pattern="/xml*" path="/" application="xmlfile"/>
```

When portal and applications are initialized, `Dispatcher` creates one instance of the `GenericWrapper` application that stores the reference to the application code. The main reason for this is to be able to assign current application stylesheets and path specification. When the document from the URI `http://my.portal/xml/data/hello.xml` is referenced, applications with the name `xmlfile`, `GenericWrapper` and `Request` are forwarded to the `XMLWrapperApplication`, which reads document `data/hello.xml` and sends it further in as a DOM-tree.

4.3.2 JSP Applications

The developed portal prototype allows separation of document code and style, and usage of tag libraries (refer to section 4.4.2). Java Server Pages (JSP) technology is very convenient for these purposes. Pages written in JSP technology are similar to Microsoft Active Server Pages [ASP], and can additionally be represented as XML documents.

4.3.2.1 JASPER

JSP applications are executed using JSP processor (in ***Tomcat*** that is ***Jasper***). It reads and interprets source document, and out of it generates Java servlet. JSP processor, then, using `forward` method from the Servlet API, dispatches to this servlet original `request` object. JSP application specifies the way `response` is created from the `request` object using the actual protocol. The output of the dynamically generated JSP servlet will through `ServletResponse` be serialized into `OutputStream`. In the JSP application the way some events are processed can also be indicated (in JSP 1.2 these events are only `init` and `destroy`, handled by `jspInit()` and `jspDestroy()` methods). However, the most important part of a JSP application compiled by a *JSP Container* is automatically generated method `_jspService()`. Author of a JSP application cannot predefine either of these methods by using scripting elements. This is done through HttpJspPage interface (or `JspPage` interface if the underlying protocol is not HTTP). Servlet class that corresponds to the JSP application has to implement `HttpJspPage` interface (or to inherit the class that implements it).

JSP Container creates one JSP servlet implementation class for each JSP application. The name of this class is implementation specific. That automatically generated servlet belongs to the implementation-dependent named package. For different JSPs different packages can be used, thus all those servlets have to satisfy some basic requirements. As already indicated, *JSP Container* executes creation of the servlet for the given JSP application. However, this process may include the superclass that the JSP application author indicates using jsp directive `extends`. *JSP Container* must check whether the superclass implements `HttpJspPage` and if all the methods of the servlet interface are declared as `final`; at the other side, the responsibility of the JSP application author is to provide that the superclass satisfies the following requirements [JSP]:

- Method `service()` from Servlet API calls `_jspService()` method.
- Method `init(ServletConfig)` stores configuration data, provides access to them using `getServletConfig()` method, and then calls `jspInit`.
- Method `destroy()` calls `jspDestroy()`.

If *JSP Container* detects that the provided superclass does not fit these requirements, it can give 'fatal translation error', but most JSP containers will not check them.

4.3.2.2 REQUESTING JAVA SERVER PAGE

Class `JSPWrapperApplication` is responsible for calling JSP applications in the portal. The request reaches the application in already described way (refer to section 4.2.4). Afrter that the following actions are performed:

- First, from `portalContext` an appropriate `RequestDispatcher` is acquired; which will transfer control over request processing to the wrapper class (for JSP it is `org.apache.jasper.servlet.JspServlet` class, i.e. *JSP* container).

- Then, `PortalRequest` and `PortalResponse` objects are created, so that the *Servlet Container* is informed where to find the page requested. Default page is `index.jsp`. Attributes `Session` and `User` are set in the newly created request, and can further on be accessed from the base class `PortalJspPage` (in the portal it implements `HttpJspPage` interface; refer to section 4.3.2.1 Jasper); thus, they are available as implicit objects to a page.
- Newly created request is then returned to the *Servlet Container,* which `forwards` it to the *JSP* container. The latter, is then, responsible for the page compilation) and forwarding request to the servlet, obtained by JSP application compilation. This servlet sets the generated result into the `portalResponse` object and that way provides it to `JSPWrapperApplication`.
- If the generated response does not contain errors, it is accepted as the result of JSP application execution, and output buffer content is forwarded to the XML parser. The latter generates complete DOM `Document`, which is then inserted into `DocumentFragment` and returned to the `Dispatcher`.

JSP applications in the portal as a result generate only the intermediate representation in XML format. This output is, then, accepted by the `JSPWrapperApplication` and, as a DOM tree, forwarded to the `Dispatcher` for further processing and transformation. The development of one JSP portal application is illustrated by the simple `HelloWorld` example. Code example 4.6 shows JSP application, which generates same output as the application 4.4. Hence, the same XSLT stylesheet 4 5 will be used.

Example 4.6 Hello World as JSP

```
<?xml version="1.0" ?>
<%@ page extends="portal.framework.PortalJspPage" %>
<%@ page contentType="text/xml" %>
<hellotext>Hello World<hellotext>
```

When deploying JSP applications the author should have in mind that:

- XML is generated as an output, so the `XML Header` has to be the part of that output. JSP directive `contentType` specifies corresponding `MIME-Type`. This directive does not have to be used in JSP portal applications, since only XML output is allowed. In the example it is indicated as an illustration.
- JSP applications, used in the portal should inherit `portal.framework.PortalJspPage` class. This way implicit usage of `User`, `Session` and other context objects is enabled.

The code example 4.6 (unlike its output) is not XML compliant. This is the reason why the automated syntax checking, using DTD, has to be provided. Code example 4.7 shows the same servlet written using XML syntax. Then both `DOCTYPE` and output type element do not have to be specified.

Example 4.7 Hello World as JSP with XML syntax

```
<?xml version="1.0" ?>
<jsp:root xmlns:jsp="http://java.sun.com/dtd/jsp_1_2">
<jsp:directive.page extends="portal.framework.PortalJspPage" />
<jsp:directive.page contentType="text/xml" />
<hellotext>Hello World<hellotext>
</jsp:root>
```

4.3.2.3 Programming Logic

The `User` and `Session` objects are, in the page, implicitly available and can be accessed using implicit variables `portalUser` and `portalSession`. Essentially, as it is obvious from the example 4.8, code can reside even inside the page.

Example 4.8 embedding Java code in JSP

```
<color>
<% Parameter p = portalUser.getParameter("myApp","Style",
   "favoriteColor");
   if (p!=null)
      write("Your favorite color: " !+p.getValue()); %>
</color>
```

4.4 Extensions

The process of application development, shown in the pervious section functions well only if the code parts are small; XML output is usually not changed by layout designer, and not frequently modified. Thus, if it is possible, the author's goal should be to store program logic in external components. In this manner, the access to external data sources is simplified. Two basic alternatives are offered in JSP: ***Java Beans*** and ***Tag Libraries*** (figure 4.14), which will, further on, be described in more detail.

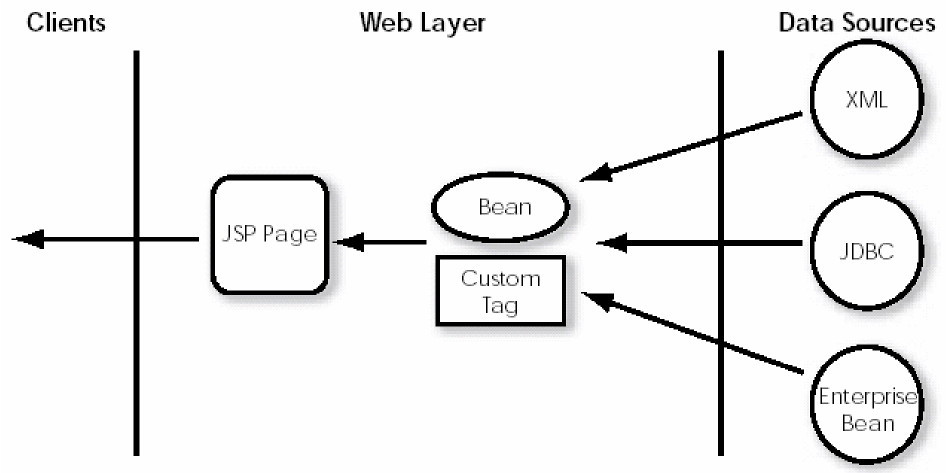


Figure 4.14 Accessing various data sources from JSP application

4.4.1 Java Beans

Java Beans are independent Java classes that are, inside the *Servlet Container*, referred to as components, which can handle requests. They are a part of ***Java 2 Enterprise Edition*** and are defined by Enterprise Java Beans Specification [J2EE]. How JSP calls on a `Bean` object can be read in JSP specification [JSP]. Source code 4.9 illustrates one possible bean usage inside the portal.

Example 4.9 Accessing Java Bean from JSP

```
<nameoutput>
<jsp:useBean id="u" scope="page"
   class="portal.beans.User"
   type="portal.beans.User" />
your name is: <jsp:getProperty name="u" property="name"/>
</nameoutput>
```

4.4.2 Tag Libraries

The other possibility for component development in JSP are tag libraries. The code is, then, executed by calling earlier defined tags. Each of those tags is implemented as Java class. Groups of tags, which have similar functionalities, are defined in the suitable tag library.

The advantage that authors get when using tags is primarily, the simplicity of their usage. Unlike ***Java Beans*** technology, tag implementation enables access to context and page elements. When using *custom tags* complex operations can be reduced to a significantly simpler form than with *beans*. Then, *custom tags* require much more initial work than *beans*; and finally, *beans* are often defined in one and later on used in different servlets and JSP applications, while *custom tags*-usually define more self-contained behavior [HALL00]. Thus, *custom tag* usage is recommended when significant amount of information between the code and the page is transferred.

In order to use *custom JSP tags* it is necessary to define three separate components:

- *tag handler* class that defines tag behavior,
- *tag library descriptor* (TLD), file that maps the XML element names to the tag implementations, and
- JSP file that uses the tag library.

4.4.2.1 TAG LIBRARY DESCRIPTOR (TLD)

The description of the tag library is stored in XML document named *Tag Library Descriptor* (TLD). This file is used by a *JSP Container* to interpret tags, from the tag library it describes. TLD is, also, used by CASE tools for JSP creation, that work with tag libraries; as well as authors, who manually create JSP applications. TLD contains some fixed information on the tag library as a whole, as well as on each of the tags (`<name/>,<tagclass/>,…`) information on *JSP Container* (`<jspversion/>`), and tag library (`<tlibversion/>`) version and information on all the actions defined in that library [JSP].

Example 4.10 A part of portal.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC...
<taglib>
   <tlibversion>1.0</tlibversion>
   <jspversion>1.2</jspversion>
   <shortname>portal</shortname>
   ...
      <tag>
         <name>list</name>
         <tagclass>portal.taglib.ListTag</tagclass>
         <bodycontent>JSP</bodycontent>
         <info>A tag that generates a dropdown from a list </info>
         <attribute>
            <name>name</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
         </attribute>
         <attribute>
            <name>defaultSelection</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
         </attribute>
      </tag>
   ...
</taglib>
```

In the code example 4.10 a part of the TLD for tag library *portal* is shown. It defines `<list/>` tag, which function and usage are described in the next section. The meaning of specific tags can be found in [HALL00]. In order to make tag library accessible to the portal, it has to be indicated in the configuration file of the *PortalServlet*, `web.xml`. For example, the description of tag library ***advisor*** is shown by example 4.11.

Example 4.11 Description of the tag library in web.xml

```
<taglib>
   <taglib-uri>
      advisor
   </taglib-uri>
   <taglib-location>
      /WEB-INF/advisor.tld
   </taglib-location>
</taglib>
```

To make the tags from the tags from tag library available to a Portal JSP application it has to be specified using, either directive:

```
<%@taglib uri="http://jakarta.apache.org/taglibs/utility" prefix="util"%>
```

by indicating suitable URI and prefix, or using namespace, for JSP application written in XML-syntax, i.e. in the following way:

```
<jsp:root
xmlns:jsp="http://java.sun.com/dtd/jsp_1_2"
xmlns:user=" http://jakarta.apache.org/taglibs/utility">
```

4.4.2.2 Tag Handler Class

When defining a new tag, first task is to define Java class that tells the system what to do when it sees the tag. This class must implement `javax.servlet.jsp.TagSupport.Tag` interface. This is usually accomplished by extending the `TagSupport` or `BodyTagSupport` class. In the code example 4.12 class handler for indicated `<list/>` tag, is shown.

Class `ListTag` inherits `BodyTagSupport` class; this means that it can manipulate its body (`<list>body</list>`). The most important part of the processing is executes method `doAfterBody()`, which returns `SKIP_BODY` value, showing that no further processing is needed. The portal application `ChooseColor.xjsp` (shown in example 4.13) uses this tag to offer identified user selection of background color in the dropdown list. Since the indicated code produces XML output, an XSLT document, needed to format that output into an HTML page, is shown in the extension (`MyDropDown.xsl`). Figure 4.15 shows one view on that web application in browser. In this example, the colors (data) are given in static XML format; so the use of this tag is not completely justified (it would be must simpler to change it with HTML `<select/>` tag).

Example 4.12 ListTag.java

```
package portal.taglib;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.BodyContent;
import javax.servlet.jsp.tagext.BodyTagSupport;
public class ListTag extends BodyTagSupport {
   private String name;
   private String defaultSelection;
   private StringBuffer buffer;
   public void setName(String name) {
      this.name = name;
   }
   public void setDefaultSelection(String defaultSelection) {
      this.defaultSelection = defaultSelection;
   }
   public int doAfterBody() throws JspException {
      String body = bodyContent.getString();
      bodyContent.clearBody();
      body =resetSelection(body);
      if (defaultSelection!=null){
      int indexOfCon= body.indexOf(defaultSelection)+
         defaultSelection.length();
         if (indexOfCon - defaultSelection.length()>=0) {
            body = body.substring(0, indexOfCon+1) +
               " selected='selected'" +
               body.substring(indexOfCon+1);
         }
      }
      if (body.indexOf(name==-1))
      buffer = new StringBuffer(
         "<" + name + "> " + body + "</" + name + "> ");
      bodyContent.print(buffer.toString());
      bodyContent.writeOut(getPreviousOut());
      return SKIP_BODY;
   }
}
```

But, even this use gives the designer total freedom to show information, since just by exchanging the `<select/>` form in XSL template; can give radio buttons instead of dropdown.

Example 4.13 ChooseColor.xjsp

```
<%@ taglib uri='portal' prefix='portal' %>
<%@taglib
 uri="http://jakarta.apache.org/taglibs/utility" prefix="util"%>
<%@page import="portal.personalization.*, portal.framework.*" %>
<util:If predicate="<% request.getParameter("color")!=null %>" >
<%
portalUser.setParameter
("JSPHello", "Style", "bgColor", request.getParameter("color"));
String backColor = portalUser == null?"red":
portalUser.getParameter("JSPHello", "Style", "bgColor").getValue(); %>
</util:If>
<portal:list name="choosecolor" defaultSelection='<%= backColor%>'>
   <color name="red">red</color>
   <color name="blue">blue</color>
   <color name="green" >green</color>
   <color name="yellow">yellow</color>
   <color name="orange">orange</color>
</portal:list>
```

b MyDropDown.xsl

```
<?xml version="1.0"?>
...
<xsl:param name="Style.bgColor" select="string('#CC3366')"/>
<xsl:template match="/">
   <html>
      <head>
      <title>Choose Background Color</title>
      </head>
      <body bgcolor="{$Style.bgColor}" text="orange">
      <form name="form1" method="get" action="ChooseColor.xjsp">
      <strong>Choose color for your background</strong>
      <select name="color" size="1">
         <xsl:attribute
         name="onChange">this.form.submit()</xsl:attribute>
         <xsl:apply-templates select="*" />
         </select>
      </form>
      </body>
   </html>
</xsl:template>
<xsl:template match="choosecolor">
   <xsl:for-each select="color">
      <option>
         <xsl:attribute name="value">
            <xsl:value-of select="."/>
         </xsl:attribute>
         <xsl:if test="@selected = 'selected' ">
            <xsl:attribute name="selected">selected</xsl:attribute>
         </xsl:if>
         <xsl:value-of select="@name"/>
      </option>
   </xsl:for-each>
   </xsl:template>
</xsl:stylesheet>
```

It is, also, possible to modify `ChooseColor.xjsp`, in such manner that the data generated in the `list` tag body is generated dynamically; for example, by inserting scriptlet that reads data from the database or generates them some other way. The main reason for this possibility is that *Servlet Containers*, before returning the tag to its handlers, evaluate the content of the tag body. In this manner, previous example, instead of static XML list, can contain the following lines:

```
<portal:list name="choosecolor" defaultSelection='<%= backColor%>'>
   <%= generateColors(10) %>
</portal:list>
```

where method `generateColors(int n)`, for example, generates list of n elements with random color distribution. To evaluate tag body attribute `<bodycontent/>` in the tld tag description, has to be set on `JSP`.

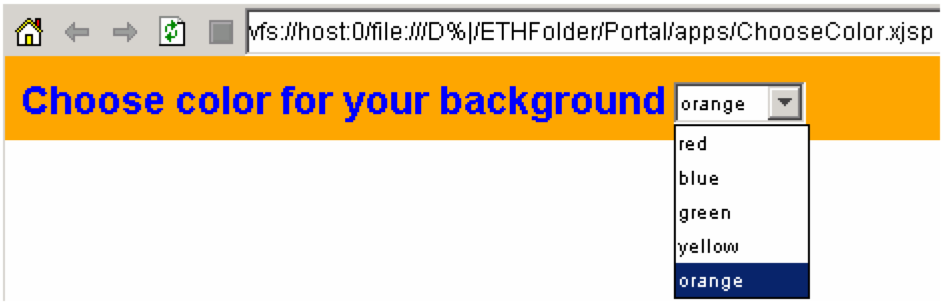

vfs://host:0/file:///D%|/ETHFolder/Portal/apps/ChooseColor.xjsp
Choose color for your background
orange
red
blue
green
yellow
orange

Figure 4.15 One view on the `ChooseColor.xjsp` application

I *BodyContent*

Variable `bodyContent` is implicit variable, containing the body of the tag, in a string format, that can be manipulated. It is an instance of the `BodyContent` class, which is a buffered writer. This buffer can be used to manipulate a tag's body content in any fashion. The `BodyContent` class extends `JspWriter`, which not coincidentally, is the type of the implicit `out` variable, which writes directly to response stream. *Servlet Container*s maintain a stack of `BodyContent` objects so that a nested tag does not overwrite the body content of one of its ancestor tags. Each `BodyContent` object maintains a reference to the buffered writer, which is either body content or the implicit `out` variable, underneath it on that stack. That writer is known as *previous out*, and is referenced by calling method `BodyTagSupport.getPreviousOut()`). The method of functioning of this stack with *BodyContent* objects is described in detail in [GEAR01].

4.4.2.3 Nested Tags

As already indicated the tags have access to the context of the page, they are part of; therefore, by storing the objects in a particular scope, tags are enabled to communicate among themselves. Communication is executed through the page context. Nested tags can do the same, but they can also communicate directly with the static `findAncestorWithClass(Tag, Class)` method from `TagSupport` class. This method, for an instance of `Tag` class (usually `this`) and class name (`Class`), which instance is an ancestor of the tag, finds that ancestor in the page context. The consequence of this fact is that nested tags can, through page context, directly communicate with their ancestors.

4.4.2.4 DBTAGS TAG LIBRARY

DBTags *custom* tag library is a part of the ***Jakarta Taglibs*** project [JTL] and contains tags that can be used for reading and writing in SQL database. It requires *Servlet Container,* which is compliant to JSP 1.2 specification. It supports utilization of `DataSource` objects, which are not part of J2SE; therefore, it is necessary, in order to use them to have J2EE or Optional API for JDBC 2.0 installation. It does not support transactions. Although this tag library is not complete, here will be explained shortly some of the frequently used tags. This is due to utilization of this library in the applications, deployed for ***ETH World*** portal prototype, and the fact that tag `CustomEventListTag` from the ***organizer*** tag library (refer to section 5.4) extends tag `<preparedStatement/>` (for details reference [DBTAG]). Each of these tags actually wraps the call to some of the classes from `java.sql` package.

I Tag <connection/>

Tag `<connection/>` accepts database URI, which can be used to connect to the (`Connection`) through `DriverManager`:

II Tags <statement/> and <preparedStatement/>

There are two ways to query database, by using `<statement/>`, or `<preparedStatement/>` tag. To query database, it is necessary to open `<statement>` tag, pass it SQL query (`<query/>`) and then:

- execute (`<execute/>`) statement, if writing, updating or deleting from database is performed, or
- call `<resultSet/>` tag, to iterate through results of the executed statement.

Tag `<preparedStatement/>` is, in a way, sophisticated format for SQL query generation. Instead of putting values directly into SQL statement, it is possible to write '?' symbols on the places where the values should be put, and then use particular tag group for setting concrete values.

III Tag <resultSet/>

The `ResultSets` are products of the `SELECT` statement execution. Tag `<resultSet/>` automatically goes through each tuple of the result set. To access each of the tuple's attribute value `<getColumn/>` tag is used; it either shows or saves the value of the attribute in the `String` format. It is also possible, to save all the results in one `ResultSet` object, under name specified by the `id` tag attribute in the page context. The latter is possible when the attribute `loop`, which standard value `true` indicates that the tag body should be executed once for each `ResultSet` row (tuple), is set to `false`.

Utilization of these tags is completely intuitive (refer to code example 5.17 `AddEvent.xjsp`). However, all these functionalities, and a few more are now covered by the standard edition of ***Jakarta Taglib*** project ([JSTL]).

4.4.3 JDOM

JDOM provides a complete *light-weight* view of an XML document, as well as its behavior. JDOM includes standard input/output behavior suitable for creating a JDOM `Document` object from existing XML document and writing a `Document`'s data to any specified location. Although the main objective of this API is to solve the deficiencies, widely recognized in SAX and DOM API, JDOM is not based on either of them. When using JDOM, author manipulates the XML document, which has a tree structure. As it is adjusted to the Java programming language, JDOM does not provide all the idiosyncrasies of DOM, but allows very quick parsing and output. JDOM is comprised of concrete (and not abstract) classes for accessing element collections, attributes and other JDOM constructions.

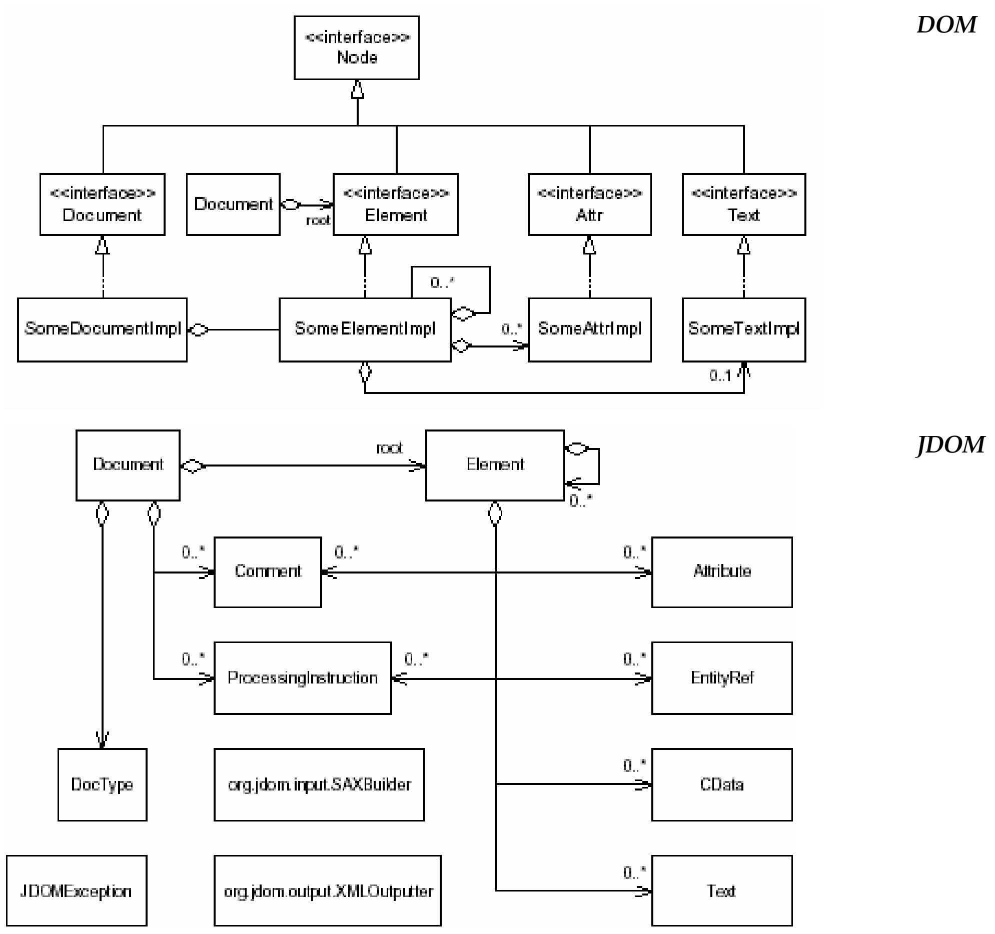


Figure 4.16 Comparing class diagrams for DOM and JDOM

Figure 4.16 shows class diagrams for DOM and JDOM. JDOM `Document` structure does not provide the strict tree representation as XML tree. This is the consequence of not supporting the `Node` interface. JDOM accesses directly each tree element. Namely, when an element is accessed the `Element` class is used, and when an attribute is accessed class `Attribute.` Unlike this, DOM, always first uses `Node` class; thus additional *cast* to the appropriate class is required. It also does not use `NamedNodeMap`, `NodeList` od `Attributes` classes to access element collections, instead standard Java classes `List` and `Map`, …, are used (see [MCL00] for details). All these features make JDOM significantly faster and simpler to use, and the cost is paid by not being compliant to standards. The code example 4.14 shows example from the example 4.1 where DOM is replaced by JDOM, which illustrates most of the differences indicated.

Example 4.14 JDOMParser

```
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;
import java.util.*;
...
public static void main(String args[]){
   Document configDoc = null;
   try {
      SAXBuilder builder = new SAXBuilder(false);
      configDoc = builder.build( "configuration.xml");
      Element
      dbconn=configDoc.getRootElement().getChild("dbconnection");
      List attributes = dbconn.getAttributes();
      Iterator i=attributes.iterator();
      while (i.hasNext()){
         Attribute currentAttr = (Attribute)i.next();
         String name = currentAttr.getName();
         dbconn.addContent(new Ele-
         ment(name).addContent(currentAttr.getValue()));
      }
      attributes.removeAll(attributes);
      XMLOutputter xmlout= new XMLOutputter();
      xmlout.output(configDoc.getRootElement(),"configuration.xml");
   }catch (JDOMException e) {
      System.out.println("readConfiguration: " + e.getMessage());
   }
   }catch(java.io.IOException ioe){
      ioe.printStackTrace();
   }
}
```

JDOM documents can be created from many different sources. For these purposes classes from the `org.jdom.input` package are used. They generate JDOM `Document` from different source formats (`File, URL, InputStream, InputSource`…). All these classes need to implement methods from the `Build` interface. Currently, JDOM provides two implementations, `SAXBuilder` and `DOMBuilder`. These allow current standard-based SAX and DOM parsers to be used for creating JDOM `Document` objects. Neither of those parsers has to provide additional JDOM support for their current offerings. However, when selecting parser one should have in mind that all current DOM parser implementations use SAX to create a DOM tree, since it is much faster. For this reason, using `DOMBuilder` to create a JDOM `Document` object does not make much sense, except in the situations, when it is created from the existing DOM tree.

4.5 Cocoon

Neither JSP 1.0, nor JSP 1.1 implementations brought expected automation of the HTML page modification process; even though the separation of concerns into content, style, and programming logic was the part of both specifications. This requirement started to satisfy applications written in JSP 1.2, which allow `Bean` and tag library utilization (for example, ***ETH World*** portal framework). First "product" that appeared on the market solving this problem is ***Cocoon 2.0 publishing framework*** [CO02]. Although there exists similarity in concepts, XSP and JSP are still not completely compatible technologies (figure 4.17).

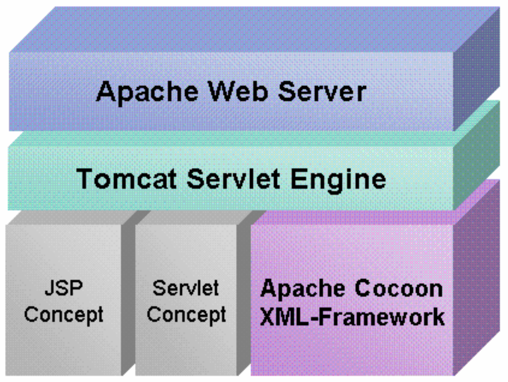


Figure 4.17 Cocoon and JSP

Web Publishing Framework. As the web server is responsible, for returning the file requested with specific URI, web publishing framework should, on a similar request, respond with the published version of that file. Published file denotes a file that is previously, probably, transformed using XSLT, changed in the application or converted in some other format (PDF, SVG...). User does not see raw data (i.e. XML document), used to produce published result, nor explicitly requests publishing current document [MCL00]. Basic criteria that this framework should satisfy are: stability and platform independence, simple integration with other XML tools, and APIs.

Unlike ***ETH World*** Portal framework, Cocoon does not have fixed dataflow in the central component, but allows pipeline definition. XML document is passed through the existing *pipeline*, through several levels of document transformation. Each *pipeline* starts with the *generator*, is continued by one or more *transformers* and finished with a *serializer* (figure 4.18). Besides normal processing, each *pipeline* can define its error handling. Additionally, it is possible to use *matchers* and *selectors*, for selecting the appropriate pipeline. Aggregation enables building the *pipeline* hierarchy. Utilization of *views* enables definition of output point for the *pipeline*. The content of XML document is represented as stream of SAX events.

Pipeline for each URI is defined in `sitemap.xmap` file, and can look like this, for example:

```
<map:pipeline>
   <map:match pattern="samples/helloworld.xml">
      <map:generate type="file" src="samples/helloworld.xml"/>
      <map:transform type="xslt" src="samples/helloworld.xsl"/>
      <map:serialize/>
   </map:match>
</map:pipeline>
```

where reference to `http://mydomain.org/samples/helloworld.xsl` starts the generation of SAX representation of the `helloworld.xml` file, which is placed in `mydomain_context_path/samples` directory; then, its transformation, using `hello-world.xsl`, situated in the same directory, and finally serialization into `HTML` format, to show it in the default browser.

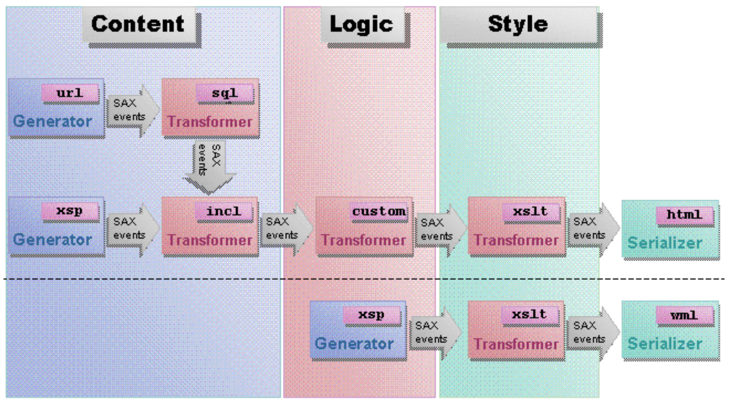


Figure 4.18 Cocoon pipeline

This is off course, the simplest possible example. As suggested Cocoon is much more complex, where this complex programming logic is executed by eXtensible Server Pages (XSP), that allow clear definition of logic-content boundaries (see [CO02]).

4.6 Commercial Portals

Currently, at the market there are tons of commercial products that help programmers in the portal development process. These products offer development environments with preprogrammed portal components that together with contents, *custom* web applications, and *back-end* systems, can be adapted and combined into final product. The portal product market has matured significantly since its birth in 1998. In the early days of this market, "pure-play" vendors were the only path to a portal product. Subsequently, middle-tier vendors and large independent software vendors (ISVs) entered the market in force. As the figure 4.19 shows, last few years the portal product market is constantly changed, and there is a major shift in market momentum.

Figure 4.19 contains the portal product Magic Quadrants for last three years [GG00-02]. "x" axis is *completeness of vision* axis, which is estimated on the basis of the complete vision of the vendor and not based on the quality of the currently available product. "y" axis measures the ability to execute, and not the way it is actually executed. Both measures are related to the future that product offers, and not to current situation. Four different vendor groups can be recognized. Leaders have good technical solutions and very convincing visions. Among vendors from this group SAP (TreeTier) and IBM are leading. Challengers are the vendors that offer products that are too concentrated on their own underlying technology, which does not actually follow the vision of the generic portal. Visionaries have good ideas, but lack stability and audience to have better results on "ability to execute" axis. Niche players are vendors that have strong visions in other areas, while portals are just their additional products.

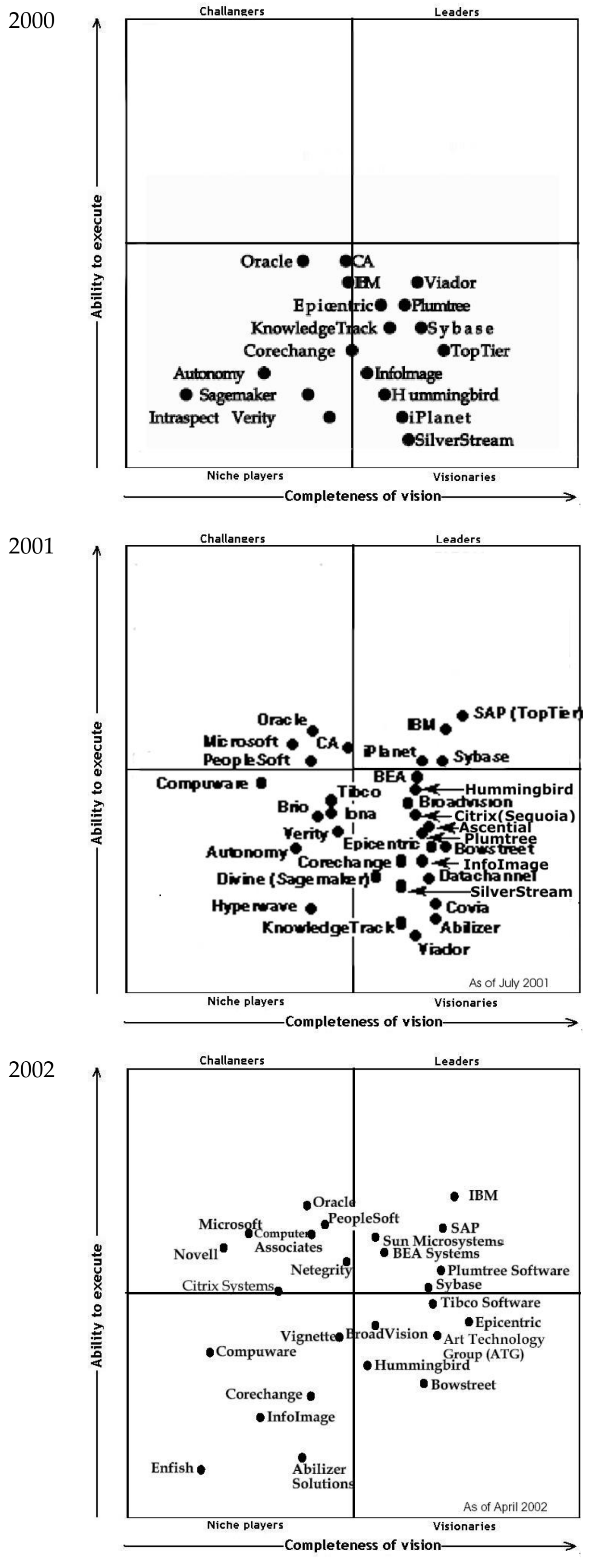


Figure 4.19 Magic quadrants of portal producers

Having in mind big changes that are happening over years on the portal products market, in this thesis just IBM and SAP portals will be shortly represented, since according to Gartner group reports, they have been altering on the leader position, and are very expensive products. At the end as some kind of parallel to the ***ETH World*** portal framework prototype a short introduction to ***uPortal***, as a joint initiative from about twenty American universities to make a campus *pocket* edition will be made.

4.6.1 IBM WebSphere Portal

WebSphere Portal Server provides open, flexible and scalable infrastructure for creation and development of different portal categories that can be accessed from numerous desktop and mobile devices. Figure 4.20 shows complete architecture of the ***WebSphere*** portal solution. Portal clients access portal over HTTP protocol, directly or through corresponding proxy servers or gateways (for example, WAP gateway or voice gateway). When aggregating pages for portal users, the portal invokes all portlets that belong to a user's page through the *Portlet API*. There are two basic portlet types:

- *Local Portlets* run on the portal server itself. They are deployed by installing *Portlet Archive* files on portal servers and are invoked by the portal server directly through local method calls. As local portlets run on the portal server itself, they provide minimal latency times. However, installing portlets usually requires assurance that the portlets are not erroneous or even malicious.
- *Remote Portlets* run as web services on remote servers. They are published as web services in a ***Universal Description, Discovery and Integration*** (UDDI) directory, to be easy to find and bind to. A remote portlet web service is bound by adding a Portlet Proxy to the portal's portlet registry when an administrator finds and selects the remote portlet web service in the UDDI directory. Portlet proxies are generic local placeholders that invoke portlets located on remote servers through a ***Remote Portlet Invocation*** *(RPI)* protocol based on the ***Simple Object Access Protocol***-u *(SOAP).*

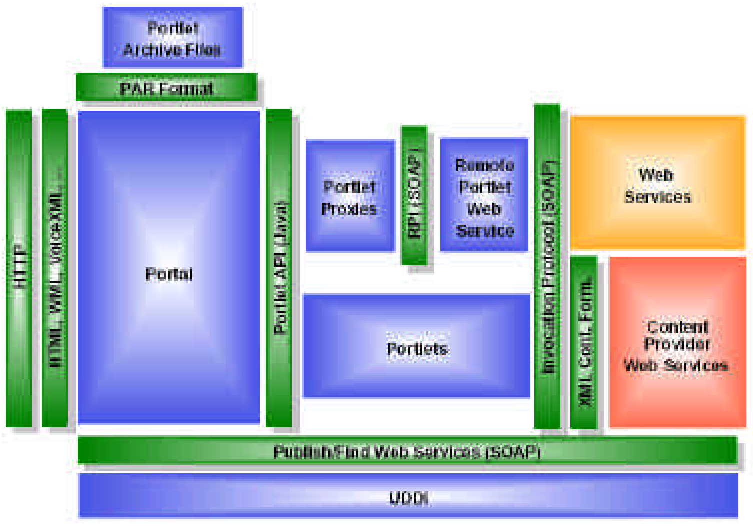


Figure 4.20 WebSphere PortalServer architecture including web services and remote portlets

4.6.1.1 Portal Engine

WebSphere Portal Server provides a pure Java portal engine, which runs on multiple hardware platforms and operating systems. The main responsibility of the portal engine is to aggregate content from different sources and to serve the assembled content to multiple devices. Portal engine separates the portal page into portlets (discrete components), which in turn enables faster, easier, and specialized development for the overall portal site. This means that the details on portlet presentation are decoupled from the ones of overall page:

- In front of the portal engine is an authentication component such as standard ***WebSphere Security***, ***WebSEAL-Lite***, or a third-party authentication proxy server.
- The central component in the portal engine is the *portal servlet*. It examines the URI and header fields of each request and invokes the appropriate handler. The portal servlet handles the request in two phases. In the first phase, portlets have an opportunity to send event messages to other portlets. (for example, portlets might send events in order to update data that will be rendered in the next phase.).
- In the second phase, the appropriate aggregation module for the user device renders multiple portlets in a single page. The aggregation modules accumulate information from each portlet, add standard decorations around the portlet (such as a title bar, edit button, and enlarge button), position it on the page, and generate the overall page markup.
- Access to portlets is controlled by checking access rights during page aggregation, page customization, and other access points such as viewing the portlet in its maximized state.

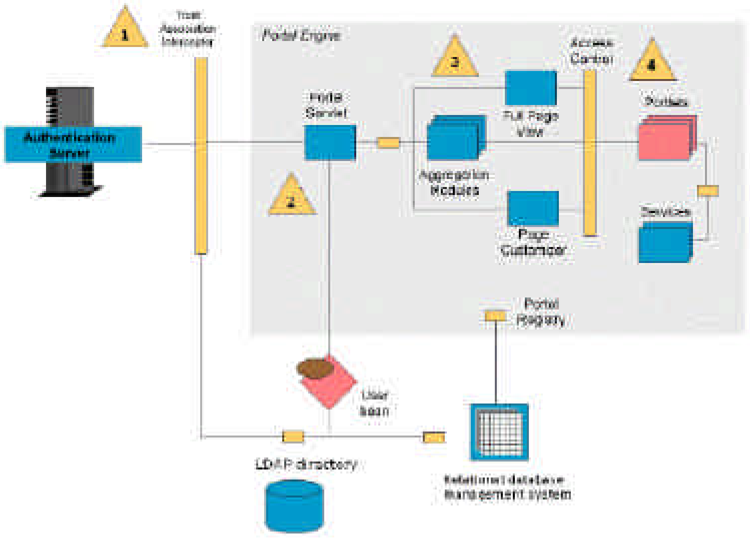


Figure 4.21 Web Sphere Portal Engine

WebSphere Portal Server currently has three aggregation modules: *HTML component* produces pages for desktop computers and other devices with HTML browsers; *WML component* produces WML content for Wireless Access Protocol (WAP) devices, such as mobile phones. *iMode aggregation component* produces cHTML markup (for mobile devices in NTT DoCoMo network).

Each user can customize a unique home page for each device, selecting the content and applications that are most useful on the device. When the home page is requested, page aggregation works by first detecting the type of device that is making the request, and then assembling the portlets which render their contents in the appropriate markup language.

4.6.1.2 PORTLETS

Portlets are visible active components that see on their portal pages. Similar to the windows on the PC desktop, each portlet covers a part of the browser, or PDA screen, where it shows its results. Portlets can be simple applications, for example e-mail, or very complex like recommending courses, which would be interesting to a particular student in the next semester.

Figure 4.22 Portlet look

Portlets are usually small applications. However, they do not give multiple outputs. Instead, portlet is often used for periodical access to application data or for placing popular information next to very important information from other applications. For the user, portlet is an information channel or application that the user can subscribe to, add it to his/her portal page and configure, so that it shows personalized content. For the *content provider*, portlet is a tool to publish his contents. For the portal administrator, portlet is a content container, which can be registered in the portal, so that the users can select it and put it onto its pages. For portal, portlet is a component that renders on some of its pages. From the technical point of view portlets are very similar to Java servlets, except that they only return a subset of the output page. Portlets use *Portlet API*, which is similar to *Servlet API*. However, portlets are executed in the portal environment, while servlets are executed as standalone applications in the *Servlet Container*. While servlets communicate directly to their clients, portlets are referenced indirectly, through portal application.

4.6.1.3 PERSONALIZATION

- Portal services are features that ***WebSphere Portal Server*** leverages to provide a complete portal solution that includes features such as personalization, search, content management, site analysis, enterprise application integration, collaboration, Web services, and so on. Since the topic of this thesis is personalization in the portal environment, in the following text there just this element is described (for detailed descriptions of all the other elements see [GAN00]). Three different personalization techniques are implemented:

- Customization is partially provided through administrative setup, which defines standard parameter values (*default settings*) and the portlet access rights, and by explicit user actions to change the content and layout of the portal *home page*.
- the rules engine uses business logic to select the contents that will be shown to the particular user;
- While recommendation engine utilizes CF technology (refer to section 3.1.4.2) to select content based on common interests or behaviors.

The portal server, the rules engine and the recommendation engine share user profile and content repositories. Recommendation engine is comprised of different tools, which utilize CF technique to analyze data, including: preference engine - tool based on the explicit user actions, transaction engine (purchase engine), click stream engine, product matching engine, and item affinity engine, which is not a CF technique, but is based on Market Basket statistic (see [GAN02]).

4.6.1.4 WEB SERVICES

The concept of web services has been developed to allow business applications to communicate and cooperate over the Internet. It allows applications to find web services in a standardized directory structure and bind to these services with minimal human interaction. Global service registries are used to promote and discover distributed services. A client that needs a particular kind of service can make a query to the global service registry to find services that suit its needs. In order to be easy to use, the mechanisms for publishing portlets as remote portlet web services, finding remote portlet web services, binding to them and using remote portlets must be integrated seamlessly into portal products (see [SCHA01] for details).

4.6.2 SAPPortal

From the figure 4.23 it is obvious that the portal infrastructure is one of the key building blocks of ***mySAP Technology for Open e-Business Integration.*** It provides user-centric collaboration. Alongside, ***mySAP*** technology [SAP01] includes: web application server, which is a base for development of specific mySAP.com solutions, exchange infrastructure that supports process-centric collaboration, and basic infrastructure services, which include: security, globalization and IT landscape management. The ***SAPPortals*** has integrated four different technologies into its portal infrastructure to access different information sources, which user needs in his/her work. These information sources include: applications and legacy databases, business intelligence, unstructured data, and web contents and services. Figure 4.24 illustrates portal server architecture. It also shows, data flow from the moment a client sends request to some information source, using the portal. It will be explained on a call to a simple ***.NET*** web application, in the case of HTML client. The Portal client's Web browser sends a request to display a web page (comprising *iViews*). The *iViewServer* calls the *iView* application through a URI. The *iView* application reads the data from the data source and processes it. The *iView* application sends the result to the *iViewServer* in HTML/XML format. For "rendered" .NET iViews, the iView application file (ASP, CGI etc.) reads the data from the data source, processes it, and sends it as ***SAP Portals*** XML (contains HRNL Tags) to the *iViewServer,* which converts it into HTML and sends to the Portal client. The Portal client displays the *iView* in the Web browser

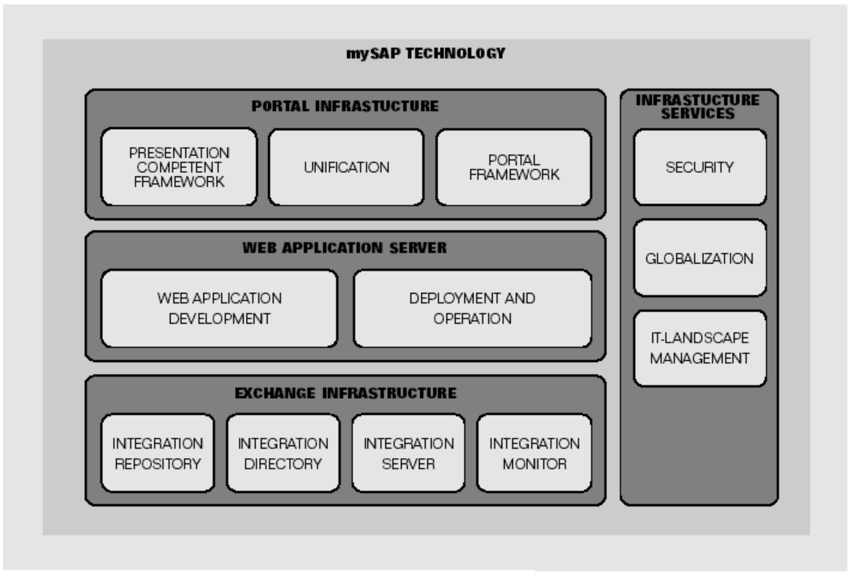


Figure 4.23 Building blocks of ***mySAP*** technology

4.6.2.1 PAGE BUILDER

The ***Page Builder*** component assembles pages, which will be rendered in the portal. It gets information about the navigation between pages and the layout of the pages from the roles assigned to a user. It gets information about the navigation between pages and the layout of the pages from the roles assigned to a user. Roles describe user's functions. Roles are assigned work sets, which in a row, consist of *iViews*. From the semantic perspective, the primary role function is to provide navigational hierarchy for pages, work sets, *iViews* and application user interfaces, while the function of each work set is the aggregation of all the important *iViews* with the goal of task execution.

4.6.2.2 IVIEW SERVER

iView Server connects to multiple information sources through ***iView Connector*** using different protocols and APIs (for example, OLE DB, HTTP, SOAP…). The *iViewServer* provides the following functions: administer *iViews* and provide their content from various sources, enable development of new content, allow import of new content such as roles and *iViews* from the *iViewStudio*, maintain cache of *iViews'* content. However, basic *iViewServer*'s function is producing the HTML content of *iViews* and pages. It also offers a delivery mechanism for web services. Communication through WSDL allows information to be passed through to the *iViewServer* from the service, and delivery of *iViews* as SOAP compliant objects permits the proper display of web service interfaces.

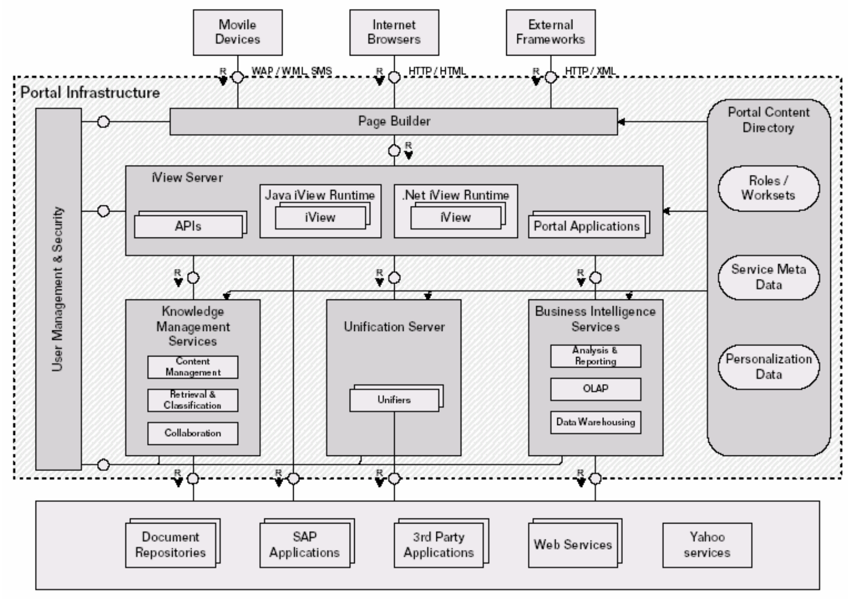


Figure 4.24 Portal Server architecture

The *iView* is a presentation Web service that aggregates content from a wide array of different applications. There are two different types of *iViews*: inherited and enhanced. Inherited *iViews* access information directly as it would appear in the accessed source, whereas enhanced *iViews* present the information in the same consistent look and feel as the portal by using a consistent XML format.

Channels fulfill two functions. First, they act as a central access point to all *iViews* available to the administrator for the construction of work sets. Second, channels are available to end users as an access point to *iViews* that are not part of a work set.

4.6.2.3 APPLICATIONS AND LEGACY DATABASES

Unification and *iViews* have been designed to access legacy and transactional systems; *iViews* provide awareness of events and enable the rapid creation of information building blocks from enterprise applications. *iViews* can be written in any programming language, yet they will generate consistent XML based output. *Unification* provides contextual navigation that greatly reduces the time it takes to resolve any event users become aware of.

IV *Unification*

Unification allows data traditionally stove-piped in various sources to be retrieved and correlated on an ad hoc basis with any other information source. This access and correlation across a variety of information sources is provided through the unification server, through its key components, application and database unifiers. Unifiers control the application architecture, user interfaces, security, and all customization inherent in an application while surfacing the application with *HyperRelational* technology (the technology that uses unified object model). The unifier is a truly transparent wrapper around the application, providing access and *Drag&Relate*™ capabilities. At the front end, the user is working from the project's *iPanel* in the Portal, and intuitively querying data sources. This consists of the user dragging visual elements representing data and dropping them onto other such elements; interrelating the data to create queries dynamically. Figure 4.25 illustrates Unification Server architecture. Each m achine containing a unifier requires a web server installation. The Unification Server is a COM -based extension to the web server.

When the user performs a *Drag &Relate* action, client sends a *HRNP* (*HyperRelationalc Navigation Protocol*) request to the Unification Server through HTTP. The Unification Server resolves the relationship between the drag source and the drop target, and queries the database to determine the record set. Then it redirects the result set to the database server and launches the screen that appears in the browser.

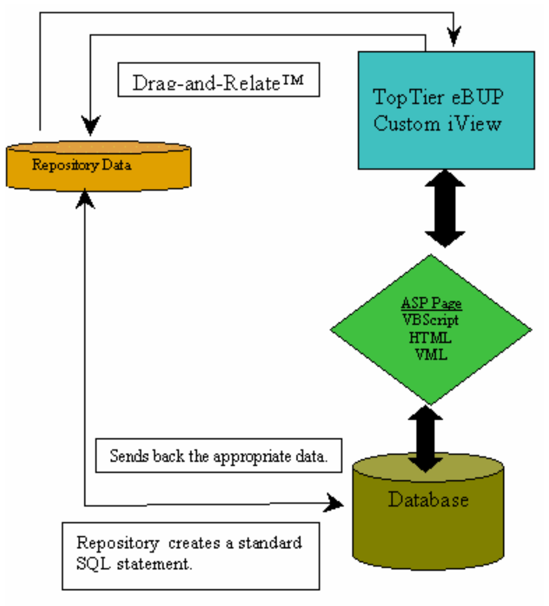

Drag-and-Relate™
TopTier eBUP Custom iView
Repository Data
ASP Page
VBScript
HTML
VML
Sends back the appropriate data.
Database
Repository creates a standard SQL statement.

Figure 4.25 Architecture of the unification server

The Unification Server features a repository of metadata, it has extracted from the database, a *HyperRelational OLE DB* provider, a security mechanism that supports *LDAP* -based directory servers (refer to section 2.5.10.1), and *ASP* applications, which running on *Microsoft Internet Information Server*, retrieve XML and display results in HTML.

V *Security*

User management and security services are integrated in such manner that single sign-on mechanism is enabled (refer to section 2.5.10.1). Since each user can have more than one role in the organization, portal grants access to important services by aggregating access rights that are defined for each role.

VI Personalization

Personalization includes just two elements: content customization and user recognition. Customization can be defined in three ways: at the administrator level, the user level, and automatically through predictive technology. Administrators can define personalization for each user by changing the design of the portal structure for different users. Users can also personalize their content through the browser, although the administrator, using the options mentioned in the previous paragraph, always controls how much they can customize their screens. Users can change the location and order of pages, change the *iView* order, and build their own role using *iViews* available through channels. Predictive technology allows for automatic personalization based on the user, their location, or the event being handled. Some of the qualities the portal can leverage include the user type, the browser type, the device type, whether the user is outside or inside firewall, and the bandwidth of their connection.

4.6.2.4 BUSINESS INTELLIGENCE

The business intelligence solution provides a complete, open, end-to-end analytics platform that can assist in aggregating information to simulate the results of current actions on future data. Standard interfaces bring data in to the data warehouse from external online transaction processing (*OLTP*) systems, applications (for instance, marketplaces), and data providers using an *extraction, transformation, and load* (ETL) process. This infrastructure also monitors and optimizes business processes and provides predefined business scenarios and key performance indicators. This information can be viewed in a variety of ways, through the portal (*SAP Business Explorer*, *iViews* combined with 3rd party BI tools, Microsoft Excel…); it can, also, be unified, which allows users to correlate the important trends in the warehouse with information available in the organization.

4.6.2.5 UNSTRUCTURED INFORMATION

Some of this information needs to be aggregated, classified, and disseminated to relevant parties within the organization. Search, version control, multiple publishing methods, and flow control of document publishing from and to the portal are also critical for knowledge management solution. The knowledge management system within the portal solution is accessed through a single interface. This interface conforms to *WebDAV* standard. This interface provides a front-end for multiple services, which administer and grant easy access to a variety of documents, through *Knowledge Management* (KM) solution. A repository within KM infrastructure stores the pointer to documents in a folder hierarchy, while versioning functions and subscription notification ensure that users are notified of the most recent document. A search-and-retrieval engine, that incorporates both simple and advanced search techniques, accesses information from both the hierarchy of documents within the repository and from any outside source defined by the administrator.

4.6.2.6 WEB CONTENTII SERVICES

Yahoo! offers an unmatched collection of content and services on the web. The portal connects to the Yahoo! servers directly, instead of scraping the pages to present information to the users. This means that as the content and structure of ***Yahoo!*** changes so will the information in the portal. It also means that any content that might be particularly prone to a change in URI, such as *iViews* that might scrape content from a ***Yahoo***! site, are unaffected by any change in the varying structure of the site.

4.6.3 UPortal

Unlike the previous two, enterprise portals, ***uPortal*** is an open-source institution information portal which is independently deployed by approximately twenty American institutions for higher education (JaSIG [JaSIG]). This group sees institutional portal as a limited and adapted version of that institution on web... *campus pocket edition*. Portal technology enriches web campus with the ability to adapt and form virtual communities. When customizing, each user defines unique and personalized view on the web campus. Community growing tools, including chat, forums, and surveys and so on, build relationships between clients and the campus. ***uPortal*** is an activity based on open standards using Java, XML, JSP and J2EE.

4.6.3.1 PRESENTATION ASSEMBLY

The primary function of the framework is to provide efficient and flexible engine for assembling a presentation. Given a set of information sources (*channels*), and a recipe on how to arrange and frame them (*stylesheets*), ***uPortal*** framework coordinates the compilation of the final document.

The starting point for any presentation assembly is always an abstract organization of channels: the `userLayout` document. The assembly process transforms the `userLayout` document in three major stages to obtain the final document in a desired markup language (figure 4.26).

The first stage translates an abstract `userLayout` hierarchy into the structural terms of the final presentation. That translation is termed *structure transformation*, and its logic is defined by the structure stylesheet. For example, the structure stylesheet for the default *tab&column* presentation will translate abstract `user layout` structure into a structure of *tab* and *column* elements. After the structure transformation, ***uPortal*** initiates rendering cycles of the channels that will be incorporated into the final presentation.

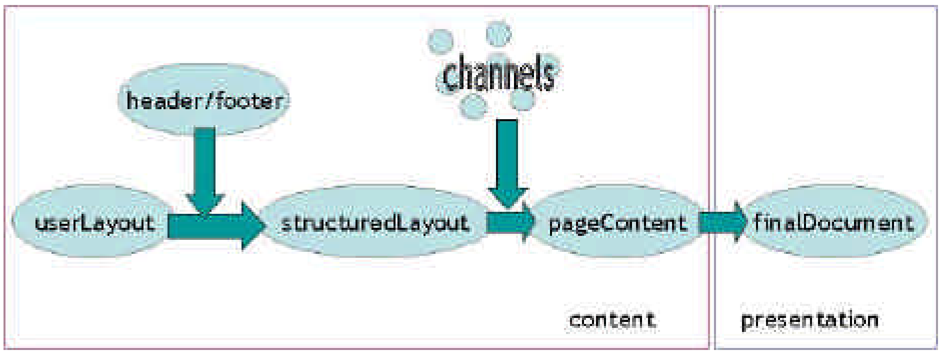


Figure 4.26 Information flow

The second stage of the assembly process will translate the result of the structure transformation into a target markup language. This translation is termed *theme transformation*, and its logic is defined by the `theme stylesheet`. For example, default `nested-tables` theme transforms a document produced by the *tab&column* structure transformation (a structure of tabs, columns, etc.) into a set of nested HTML tables that visually resemble tabs and columns. The content provided by each individual channel will be incorporated into the result of the structure transformation.

The final stage serializes the result of the theme transformation together with the channel content into a stream of characters according to the rules of the target markup language and media.

Figure 4.27 illustrates the system components that take part in the presentation assembly. An entry point into the ***uPortal*** is `PortalSessionManager`. That is a J2EE servlet, conforming to the specification version 2.3. It is responsible for identifying and managing incoming HTTP requests and dispatching them to corresponding `UserInstance` objects. `UserInstance` is an object that encompasses all of the information for an individual user. It contains `UserLayoutManager` and `ChannelManager` objects for the user's session. It also bares the responsibility of arranging the rendering pipeline. `UserLayoutManager` maintains references on `userLayout` document and `userPreference` object. It mediates any changes made to the `layout` or `preference` objects for the current user session. Object `ChannelManager` maintains instances of channels for the current session. It is responsible for distributing runtime information to the channels and generating `ChanelRenderer` objects. It also caches information on channels. `ChanelRenderer` objects are responsible for driving rendering cycles of individual channels.

4.6.3.2 CHANNELS

Channels are the unit sources of information for the ***uPortal***. The framework organizes channel behavior to produce a coordinated output of the content provided by them. ***uPortal*** framework provides channels with the means to achieve their goals, but does not try to enforce how things are done. Channels are Java objects implementing `org.jasig.portal.IChannel` interface. They are stateful entities, and spend most of their lifetime in a rendering cycle. Administrative life of a channel consists of the following phases:

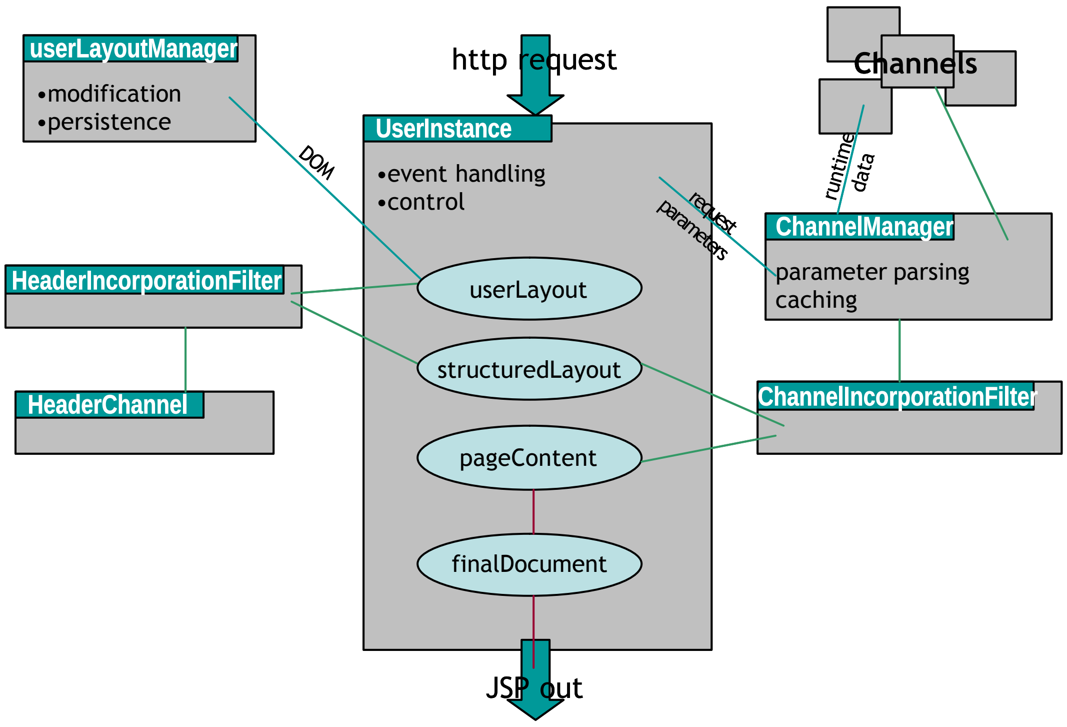


Figure 4.27 uPortal components

- *Creation:* At this phase the code for the channel class and all of the necessary resource files are created and a *channel publishing document* (CPD file) is produced. CPD files provide descriptive information about the purpose of the channel, specify the channel Java class, describe configuration parameters, and outline the workflow of the publishing and subscription process for this channel. The above tasks are usually carried out by the channel author. Upon the completion of this phase, the channel can be distributed to the ***uPortal*** installations.
- *Registration*: When an administrator of an ***uPortal*** installation acquires a new channel, the channel has to be registered with the system. During the registration process, the portal is made aware of the channel's existence, and the channel is assigned `channelTypeId`.
- *Publishing*: In order for a registered channel to become available for subscription by ***uPortal*** users, it has to go through a publishing process. During this process channel configuration is determined by assigning values to the channel's parameters. Such channel configuration is assigned a `channelPublishId`. A published channel is then placed in taxonomy of channel categories.
- *Subscription:* ***uPortal*** users can go through a subscription process to insert a published channel into their personal layouts. Subscription process finalizes channel configuration by assigning values to all of the necessary chan-

nel parameters. A subscribed channel is identified by a `channelSub-scribeId` that is unique within the scope of the user's layout.

4.6.4 Conclusions

The architectures of most of the currently available systems, shown in previous sections, indicate that the concept and architecture of ***ETH World*** portal framework do not fall behind actual world trends, and were pretty avantgarde at the time of their creation (see [JAUS01]). Although, both ***IBM WebSphere*** and ***SAPPortals*** systems offer numerous, already implemented options, the tradeoff between the price and platform independence, is still not affordable in the academic environment. One should have in mind that open architectures of both, ***uPortal*** and ***Cocoon***, *open-source* frameworks, allow almost the same possibilities, but on incomparably lower price. Thus, the prototype, described in this master thesis, relies on standard Java tag libraries, as another *open-source* solution. Although, neither ***uPortal,*** nor ***Cocoon***, at the moment of prototype implementation, supported tag libraries trends in their development, were announcing that they would; which would make this tag library available to both systems.

5 Implementation

5.1 Introduction

As it is mentioned earlier, the goal of this thesis was to introduce another element of personalization into the ***ETH World*** portal framework: recommendation, since the original portal implementation includes other two elements, customization, and one-to-one relationship to the user. The motivation lays in the fact that in university environment this facility could find many different applications; for instance it could help students in the process of course selection when they are starting new semester or find colleagues with similar interests. It can also find many applications among other members of university community, to researchers, for example, when discovering important papers, books or bookmarks and many more. This concept has shown its good qualities in business environment, especially *e-commerce* (take ***Amazon.com*** as example), so why not try it in the academic environment.

In the section 1.4, the main objective of the ***ETH World*** project to transfer the university spirit into the virtual world, is explained. Along with the basic functions of the institutional information portal, the ***ETH World*** portal need to provide something m ore; it has to become personal gateway and guide through university life. This is were the recommendations find their place, as a specific type of user help, in the navigation through virtual space and un the selection of the right in the piles of existing contents. Each semester, tons of students have to choose courses coherent to their interests and their future needs. It is a hard task and requires lots of investigating, conversation, and thinking. Contradictory opinions come from different sides and there is no guarantee that if more people are inquired the quality of deduction will grow. On the university, already exist tons of information about each member of academic community, both historical and contemporary. This information can be used for inferring new facts on them. For example, there exists information about exams passed and courses attended for every student. If we assume that chosen courses show interests of that particular student, it is possible to partition the student population into clusters with similar interests. These are the students whose *sets of courses chosen overlap*.

Similar approach is used in the ***Siteseer*** system [RUCK97]; where as interest indicators the web browser bookmarks are used. The assumption, on which this system relies, is that users bookmark interesting web sites and organize related bookmarks into folders. Figure 5.1 illustrates the approach, used in this system, on the example where *John* should get recommendations for vacation location. Based on the degree of overlap (such as common URIs) in the thematically-similar folders, the most qualified recommenders (i.e. nearest neighbors, there are four of them) are detected. Recommendations are those URIs, which have been bookmarked by the user's virtual neighbors, drown from folders with the highest overlap as well as those held within multiple folders in the neighborhood (see [RUCK97] for details).

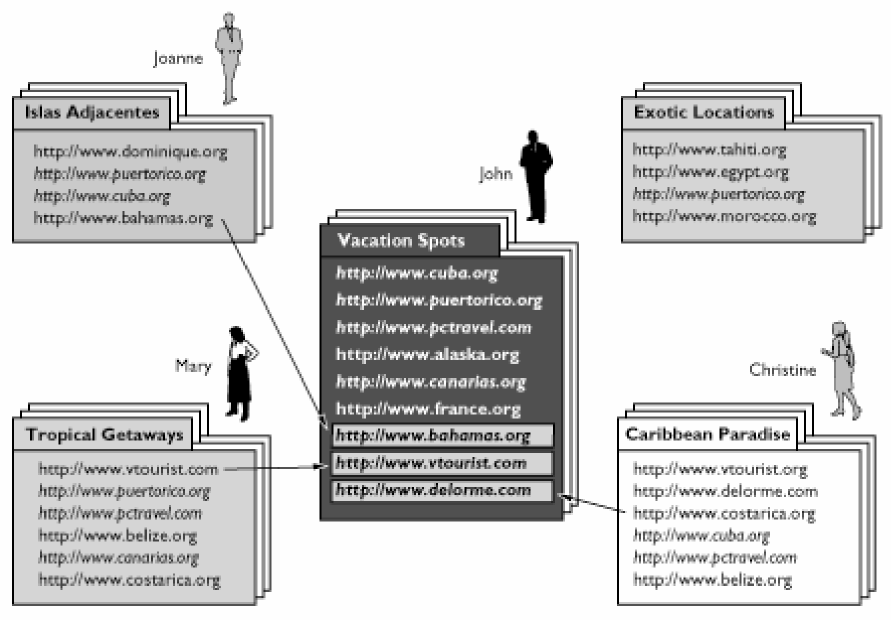


Figure 5.1 Finding virtual neighbors and recommendation in the ***Siteseer*** system

The ***ETH World*** portal can, for example, in the course selection process, show the list of courses, selected by his nearest neighbors (in the interest space). This could be mapped to a *real world* situation where the student gets advices from people that share same interests. The advantage is that the student does not have to spend precious time trying to find people that share the same professional interests, and investigate them, but it is enough to ask the portal.

I Concepts

Modeling users (actually, generating profiles) with the set of his/her interests is usual in the recommender systems (refer to third chapter). Depending on the required profile persistency and the level of user engagement, different user actions can be utilized as interest indicators. Table 3.2 shows some examples of the academic systems, which as interest indicators use followed links, specified keywords, and user ratings of already viewed contents…

In the CF systems the users are, often, represented as points in n-dimensional content space, where the interests (actually, interest indicators) represent the values of their coordinates (figure 5.2a). First are in the user space, identified points closest to the active one, and then recommendations are generated, as those dimensions of the closest points which active one does not hold (refer to previous example, figure 5.1). There is also an approach in which dimensions are users, and contents points in that n-dimensional space and the value of its coordinates are again interest indicators (figure 5.2b; description in more detail in section 3.4.4). In IR systems (figure 5.2c), also, the closest points are found, but now among contents in the interest indicator space; and they are recommendations.

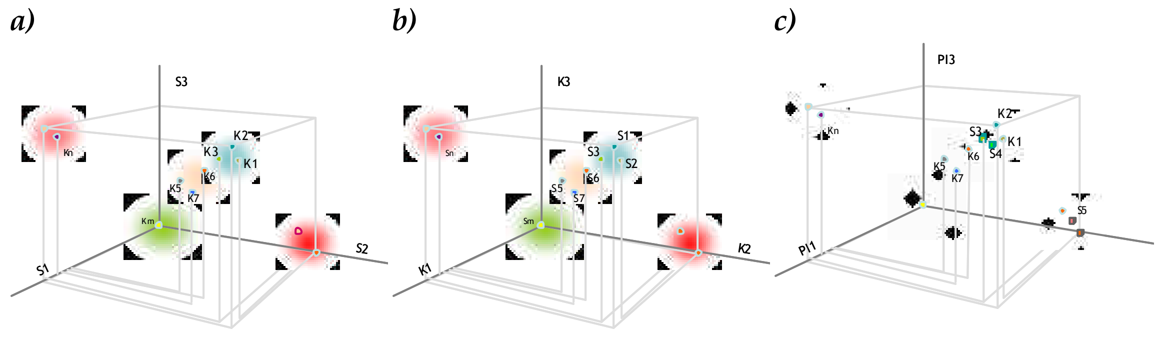


Figure 5.2 Generating recommendations by identifying: closest users(a), similarly rated contents (b), similarities between users and contents(c)

Figure 5.2 brings us to the conclusion that, actually, the principle of grouping profiles is the same, just the profiles are different (dimensions inverted, or parameterized). When the goal is the functional extension of the framework, the generic solutions are sought for. In this case, it is a module that will allow selection in both, profile definition, and the way similarity between them is resolved. This actually means that the application programmer has the freedom to choose what she/he will use as interest indicators, and different ways to collect them are provided.

One should, also, have in mind that the user profiles (user representations in the interest space and inverse; figure 5.2) change over time; and the profiles of their nearest neighbors change, too. Profiles migrate from one group to another; hence, groups change their organization. People, who once had similar interests to mine, and were my friends, may be now thousand miles away. Thus, the groups should be defined as dynamic, and for their creation use some of the existing algorithms.

II Technology

There is a number of algorithms that allow dynamic group creation i.e. profile clustering (refer to section 3.3.2), which is the name for these algorithms in the *data mining* world so the aim of this thesis was not to so make a new one. For *proof of concept* existing ones can be used as well. Therefore, the developed recommendation module is implemented in a way that it can use one of the existing *data mining* products; in this case it is ***Microsoft Analysis Services***, an MS SQL Server extension.

The recommendation module is implemented as JSP tag library(refer to section 4.4.2) named ***advisor***, that serves as a bridge between ***Microsoft Analysis Services*** that does the clustering and the portal, which uses those clusters. When the portal receives the information on user or content clusters, it uses this information to recommend contents that it considers will be interesting to that particular user. One of the main goals was to be independent of the ***ETH World*** portal framework as much as possible and hence *portable* to all other *Java-XML* frameworks, which provide user identification (for example, portals). Alas, at the moment portability cannot be so easily granted. A system that wants integrate this recommendation module has to fulfill long list of requirements, in addition to the already mentioned user identification. A data mining provider that implements ***OLE DB for Data Mining*** is a must.

The ***ETH World*** portal framework, as already explained (refer to chapter 4.3) enables simple integration and execution of ***Java Server Pages***. These applications, by being able to separate presentation and programming logic, support one of the basic portal functions to represent same data differently to different users. Additionally, since each of the tags encapsulates some functionality their use is intuitive and therefore available to non-programmers, too. Tags enable production of new or manipulation of already existing data. Using them, for example, data about certain user can be accessed; various lists can be outputted in XML format, distinct elements in a list can be found and so on. The only requirement that the portal Java Server Page has to fulfill is to output data in XML, so later it can be rendered by using XSLT.

III Applications

In order to prove that *this really works,* different small portal applications were developed. The first one generates interactive timetable, automatically when the user through the portal enters ***ETH World***. It provides the user a view to all the information it can find in the ETH, and which is connected to his or her university activities. This data does not have to be provided by the user explicitly but can also be inferred from available information. For example, for first semester Computer Science students during semester (weeks), information about schedules of their compulsory lectures and exercises will be presented. In the timetable there exist links to the sites that provide lots of information on courses, available in digital format. Additionally, they are provided the ability to customize it by adding or removing different events, which can be university connected or their own.

In some school systems students have the freedom to choose courses to attend and probably take the examination from. Therefore, they should be presented the list of subjects they are allowed to choose. There are many different ways to do that, but in this thesis just two of them are handled. Courses can be chosen from the recommendation list, which is generated by the application using the ***advisor*** module, or from the list of courses available for the direction of studies and semester the student is currently in. This application is available to all (other) users of the portal, students in higher semesters or different direction of studies, researchers, teachers, and … Depending on the group users belong to, it should provide different kind of information and degree of freedom.

Application ***Interactive timetable*** is available to all (other) users of the portal, students in higher semesters or different direction of studies, researchers, teachers, and … Depending on the group users belong to, it should provide different kind of information and degree of freedom. It was the initial idea when developing this application that it (together with the portal) can also be used as communication channel between all its users. For example, let an assistant add some information connected to the assisted course, to his or her personal timetable as public (visible to all). This information could be a document or event outside the regular lecture hours. Through the portal it will become immediately available to all students assigned to this course and not just to those who are conscious enough to check course homepage or regularly visit lectures. It could also serve as a reminder, which students do not have to set on their own. The main goal of this application deployment was to examine the other basic portal requirement: to have everything needed for work on one place, *a single point of entrance* into university community.

5.2 System overview

As it is already indicated, many different components mutually interact in order to make the system work as expected. Figure shows all the components of the system and the execution flow. When the client accesses the Java Server Page that calls tags from the ***advisor*** module, the following actions are taken:

- While interpreting the JSP application, ***Tomcat*** comes to an ***advisor*** tag, and starts its execution.
- The advisor tag calls ***JACOB*** (Java COM Bridge, refer to section 5.3) which connects to ADODB,
- Which then accesses ***Microsoft Analysis Manager*** through ***OLE DB for Data Mining,*** gets the required piece of information and sends it the same way back.

The figure 5.3 indicates that the ***advisor*** tag library is not completely independent from the ***ETH World*** portal, as it is previously mentioned. Tags frequently interact with portal framework and use the *database pooling* component, which provides access to ***Microsoft SQL Server*** database, where all the data on users is stored and shared by various applications. User identification is inherited from the portal, too.

In the following two chapters, a closer look is taken into ***JACOB*** and ***OLE DB for Data Mining***. It is essential to understand the way these components function in order to understand the way recommendation is achieved.

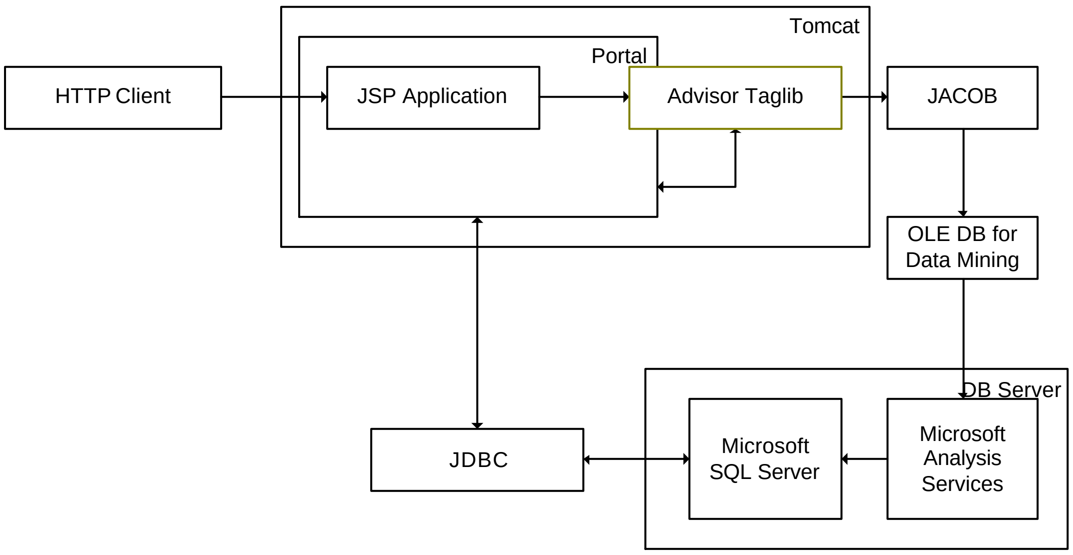


Figure 5.3 Execution of xJSP application in the portal framework

5.2.1 JACOB

JACOB (JAVA-COM Bridge) [JACOB] is a Java library that allows applications to call COM automation components from the Java environment. It uses the ***Java Native Interface*** (JNI) to make native calls into the COM and Win32 libraries. COM is the object model for creating software applications from independent components. COM objects have binary interfaces that are language-independent and therefore can be used for developing various application parts in any COM-compliant language.

Example 5.1 shows one application of JACOB, where Microsoft Excel is used as an *Automation server*, i.e. its execution is controlled by the Java application *DispatchTest*. For this reason this application is an *Automaton controller* (client). The result of the execution of this simple application, which puts values of the function $x_i = x_{i-1}^{0.9}$, $i=1,5$, $x_0=256$, in two Excel sheet rows is shown by the figure 5.4.

Application *DispatchTest* is initialised as *Single Threaded Apartment (STA).* Apartment thread is equivalent to the user-interface thread in WIN API. This means that all COM objects created in this thread are single-threaded i.e.:

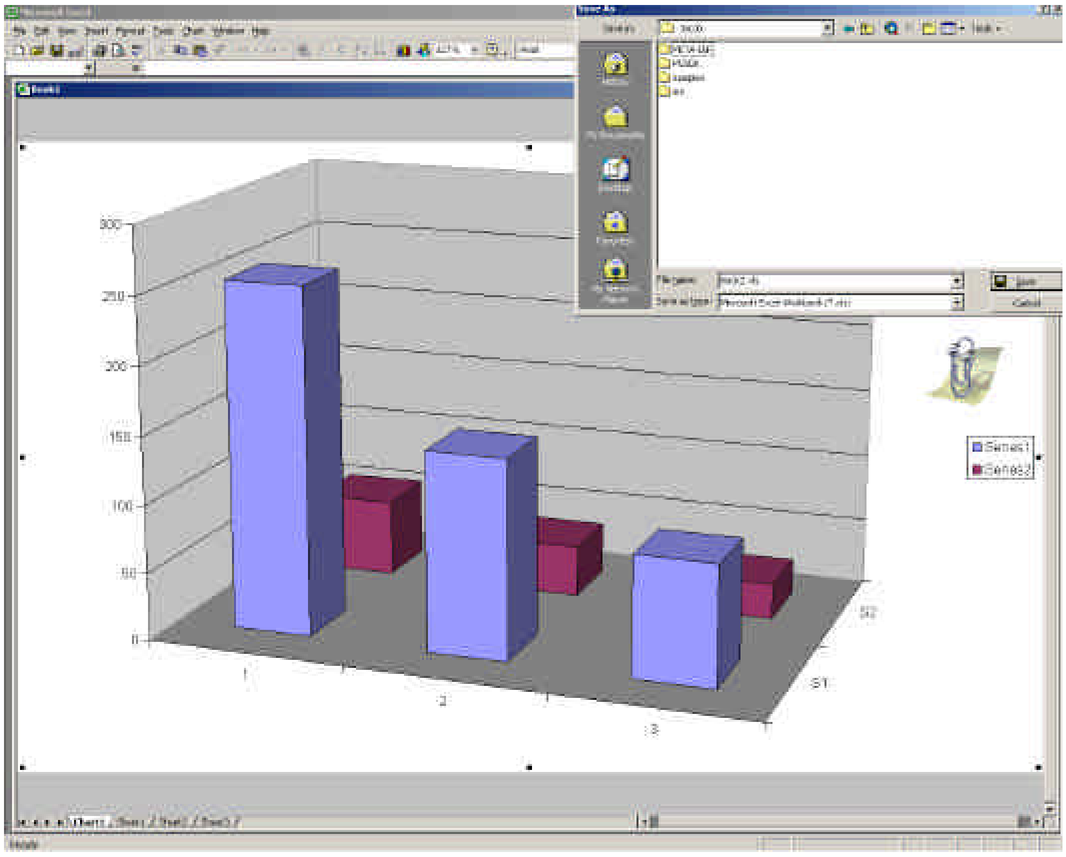
- Either all calls into that component are made from the same thread that created the component (like it is the case in the example 5.1) or

- Any call that is made from another thread gets serialized by COM, where serialization of calls is done by using a Windows message loop and posting messages to a hidden window.

Example 5.1 Microsoft Excel used as Automation server

```
import com.jacob.com.*;
import com.jacob.activeX.*;
public class DispatchTest{
   public static void main(String[] args){
   ComThread.InitSTA();
   //creates new ActiveX object
   ActiveXComponent xl = new ActiveXComponent("Excel.Application");
   Object xlo = xl.getObject();
   //defines that Excel window is visible
   xl.setProperty("Visible", new Variant(true));
   xl.setProperty("DisplayAlerts", new Variant(0));
   //creates new workbook and adds it to Excel workbooks
   //sets the work sheet and defines the fields to be used
   Object workbooks = xl.getProperty("Workbooks").toDispatch();
   Object workbook = Dispatch.get(workbooks,"Add").toDispatch();
   Object sheet = Dispatch.get(workbook,"ActiveSheet").toDispatch();
   String[] fields = new String[]{"A1","B1","C1","A2","B2","C2"};
   Vector xlfields = new Vector();
   for (int i=0;i<6;i++) {
        Object fl = Dispatch.invoke(sheet, "Range", Dispatch.Get,
        new Object[]{fields[i]}, new int[6]).toDispatch();
        xlfields.add(fl);
   }
   //sets the value of field A1 to be 256
   Dispatch.put(xlfields.elementAt(0), "Value", "256");
   // sets the values of all other fields using defined function
   for (int i=1;i<6;i++)
        Dispatch.put(xlfields.elementAt(i), "Formula",
         "=POWER("+fields[i-1]+";0.9)");
   //and then reads hem from Java application and uses (prints)
   for (int i=0;i<6;i++)
        System.out.println(fields[i]+" from excel:" +
        Dispatch.get(xlfields.elementAt(i), "Value"));
   //creates new object (chart graph) from table def by sheet object
   Object charts = xl.getProperty("Charts").toDispatch();
   Object chart = Dispatch.get(charts,"Add").toDispatch();
   Dispatch.put(chart, "ChartType", "-4100");
   Object a1c2 = Dispatch.invoke(sheet, "Range", Dispatch.Get,
   new Object[]{"A1:C2"}, new int[1]).toDispatch();
   Dispatch.invoke(chart, "SetSourceData", Dispatch.Method,
   new Object[]{a1c2, "1"}, new int[1]);
   //calls print option in Excela and prints workbook
   Dispatch.call(workbook, "PrintOut");
   //closes Excel
   Dispatch.call(workbook, "Close", new Variant(true));
   xl.invoke("Quit", new Variant[]{});
   //releases the thread
   ComThread.Release();
   }
```

a Excel output



b Console output:

```
A1  from excel:256
B1  from excel:147.03338943962
C1  from excel:89.2635946464259
A2  from excel:56.9648469281598
B2  from excel:38.0229878910691
C2  from excel:26.4266198343503
```

Figure 5.4 Using Microsoft Excel as automation server

JACOB implements the package `com.jacob.com.` All JACOB objects which wrap COM objects extend `com.jacob.com.JacobObject`. This class has some special code to register itself with a `com.jacob.com.ROT` object (*Running Object Table* -ROT). This table maps a Thread to the set of `JacobObjects` created in that thread. When `ComThread.Release()` is called the ROT checks whether that thread has created any objects, and these objects are released by calling their native `release` methods. This means that lifetime management method ties the lifecycle to the thread's lifecycle and not to the *garbage collector*. It is important to call `ComThread.Release()` on any thread before it is allowed to exit, otherwise random crashes can appear during execution.

It is noticeable that m ost of the communication between the COM component and the Java application is done through the methods of the `Dispatch` class therefore it will be explained in some more detail in the next section.

5.2.1.1 COM.JACOB.COM.DISPATCH

In JACOB, the `com.jacob.com.Dispatch` class is the basic building block. This class is used to create an instance of an Automation server. The `com.jacob.activeX.ActiveXComponent` class (example 5.1) simply extends the `Dispatch` class to provide compatibility with the way Automation servers are created in the MS JVM. The `Dispatch` class enables Java programs to invoke methods and access the properties of any ***Microsoft ActiveX Automation*** object. (ActiveX Automation objects are ActiveX components that support the `IDis-patch` interface.). All methods in the `Dispatch` class are static.

- *Constructor*: The `com.jacob.com.Dispatch` constructor takes a String as an argument. If the string contains a colon (':'), it is interpreted as a Moniker (*surrogate* COM object that holds the name, URL of the *real* object, which could be an embedded component, but usually is an Internet file e.g. "`http://host/filename`") otherwise, it is assumed to be a `ProgID` in the *Registry.* (refer to section 5.3.2).

Table 5.1 Parameters of methods invokev and invokeSubv

Object pDispatch	Either a Java object that wraps an ActiveX Automation object or a Variant object that can be coerced to such an object (e.g. chart).
String dispName	The name of the method or property that is invoked (e.g. Range).
int dispID	Indicates the method or property that is invoked. This parameter is ignored if *dispName* is null.
int lcid	Indicates the locale identifier. Wrapper methods automatically pass `Dis-patch.LOCALE_SYSTEM_DEFAULT` for this value.
int wFlags	Indicates the type of access: must be `Dispatch.Method`, `Dispatch.Get`, `Dispatch.Put` or `Dispatch.PutRef`. Most of the wrapper methods provide a fixed access that is indicated by the method name: Get methods default to `DISPATCH_PROPERTYGET`; Put methods `DISPATCH_PROPERTYPUT`; Call methods to `DISPATCH_METHOD`.
Variant vArg	The parameters of the method or property accessor. No default value
int uArgErr[]	If the Automation object rejects a parameter, this array receives the index of the offending parameter; it must be null or a one-element integer array

Although the `Dispatch` class includes a large number of methods, three basic methods define its capabilities.

- *GetIDsOfNames*: Maps methods or property names to `dispids`.
- *Invokev*: Invokes methods or accesses properties.
- *invokeSubv*: Invokes subroutines.

All the other methods in the class have the same function as one of the three basic methods; however, each of the other methods supplies (wraps) commonly used default parameters for ease of use. The table 5.1 describes the parameters required by the *invokev* and *invokeSubv*methods.

5.2.1.2 COM.JACOB.COM.VARIANT

The `Variant` class of the `com.jacob.com` package is used to bridge Java with ***Microsoft ActiveX*** components that manipulate *VARIANT* data types. *VARIANT* data type is an all-purpose data type that `Dispatch.invokev` uses to transmit parameters and return values. Most `Variant` methods fall into one of three categories:

- *toXXX* methods, which attempt to convert the `Variant` object to type XXX and return the converted value.
- *getXXX* methods, which succeed only if the `Variant` object is already the correct type. If not, a `ClassCastException` is thrown.
- *putXXX* methods, which change the type of a `Variant` object and initialize it to a new value.

Example 5.2:One application of the jacob.com.Variant class

```
import com.jacob.com.*;
class variant_test{
   public static void main(String[] args){
     System.runFinalizersOnExit(true);
     Variant v = new Variant();
     v.putInt(10);
     System.out.print("got value as int="+v.toInt());
     System.out.println(" got value as real="+v.toDouble());
     v.putString("1234.567");
     System.out.print("got value as string="+v.toString());
     System.out.println(" got value as int="+v.toInt());
     v.putBoolean(true);
     System.out.print("got value as boolean="+v.toBoolean());
     System.out.println(" got value as string="+v.toString());
     v.putBoolean(false);
     System.out.print("got value as boolean="+v.toBoolean());
     System.out.println(" got value as int="+v.toInt());
     v.putCurrency(123456789123456789L);
     System.out.println("got value as currency="+v.toCurrency());
   }
}
```

Console output:

```
got value as int=10 got value as real=10.0
got value as string=1234.567 got value as int=1235
got value as boolean=true got value as string=-1
got value as boolean=false got value as int=0
got value as currency=123456789123456789
```

Example 5.2 illustrates the application that uses one instance of the class `Variant` to store in and read from values of different types. It shows the flexibility of this class, as well as the employment of previously indicated methods.

5.2.2 OLE DB for Data Mining

OLE DB for Data Mining [OLEDB] is an OLE DB extension that supports data mining operations over OLE DB data providers. If we refer to software packages that provide data mining algorithms as *data mining providers,* and those applications that use data mining features as *data mining consumers,* then ***OLE DB for Data Mining*** specifies the API between these two.

5.2.2.1 DATA MINING MODEL

OLE DB for Data Mining introduces a new virtual object, named the data mining model (DMM) and several new commands for manipulating it. A data mining model has behavior very similar to an ordinary database table. It can be created using a `CREATE` statement, which is similar to the `SQL CREATE TABLE` statement. Models are populated using an `INSERT INTO` statement and a `SELECT` statement is used for making predictions and exploring the data mining model. OLE DB for DM treats a data mining model as a special type of table. After the data is inserted into the table, it is processed by a data mining algorithm and the resulting abstraction is saved. This means that a data mining algorithm finds patterns in training data and these patterns are saved, rather than the training data itself. An already created mining model can be browsed, refined, or used to derive predictions. The predictions are usually made on unprocessed data (data that is not used for training). That data is then processed through the model (saved abstractions). This processing is done by selecting tuples from a table, which is the result of `PREDICTION JOIN` between the data itself and the model.

The logical representation of data that will be mined is as a collection of tables in a relational database. If we take as an example a database about students, it might contain the records on students, demographic data about students, examination periods, and courses. A join of the last two may have many records for one student (one per exam seat during examination period). The collection of data related to a single entity is considered as a *case*. The set of all relevant cases is called a *case set*.

OLE DB for Data Mining uses nested tables (the way they are defined by Data Shaping Service [AHL99]) to represent these relationships. This way the columns of a model describe all of the information about a specific case. A model for the previously mentioned student example could contain the data indicated in the table 5.2.

From the table 5.2 it is obvious that a student case is not so easy to describe by using simple relational tables. The meaning and the structure of the nested tables differ from case to case. Therefore the ability of a case to contain multiple tables of data is the key requirement of a data mining algorithm. The main row of the case is the *case row*. Columns in the case row describe the entity of the case. For example, the parameter *Group* in the previous table contains the study group of a student whose *Student ID* is 1. Rows inside nested tables are *nested rows*. Columns in the nested rows describe the entity of the nested row and the way it is related to the case row. For example the mark column represents the mark that student 1 got for the *exam* in the *examination period*.

Table 5.2 Case example from the StudyLengthGuess model:

student ID	Group	se-meste r	age	mns	mns prob.	Exam	mark (5-10)	exam period	course	course prob.	ty pe
1	design	5	22	9	88%	Arts	10	SE00	web design	50%	C
						art history I	9	SE00	ind. design 3	80%	K
						ind. design I	8	WE01			
						CAD	7	WE01			

A convenient feature of nested tables, which are defined this way, is that the same physical data can be used to generate different case sets for different analysis tasks. If someone would, for example, choose to mine a model over specific courses, each course then becomes a single case and students become attributes of the case.

The example shows that each column in a model can represent one of the following content types:

- `KEY` columns are the ones that identify a row (student ID).
- `ATTRIBUTE` is a direct attribute of the case. It represents some value for the case (study group, semester, age or mark for the particular exam).
- `RELATION` represents information used for the classification of attributes, other relations and key columns. The value of the relation must be consistent for all instances of the column it describes. For example, column *type* classifies courses into: kernel*(K), credit(*C) and so on.
- `QUALIFIER` is a special value associated with an attribute that has a predefined meaning for the provider. A qualifier would for example be probability that an attribute is correct. The qualifiers are optional and apply only if the data has uncertainties attached to it or if the output of the previous predictions is being chained as input to a next training step of a data mining model. Some of the examples for the qualifiers are: `PROBABILITY, VARIANCE, SUPPORT` … Case in the table 5.2 contains attribute *mns probability*, which is a qualifier for attribute *mns;* where *mns* (*maximal number of semesters*) represents predicted study duration (in semesters) for the specific student.
- `TABLE` is a data type of a column that contains nested table. For any given case row, the value of a `TABLE` type column contains the entire contents of the associated nested table. In the `CREATE MINING MODEL` command syntax, nested tables are described by a set of columns. All these columns

are contained within the definition of a named TABLE type column (see query 5.1).

- DISCRETE: The attribute values are discrete, and usually describe categories (study group, for example), therefore often called *states*. Even if the values are numeric, no ordering is implied.
- ORDERED: Columns define ordered set of states. Even though the ordering exists terms like distance or absolute value are not defined. For example, marks (5-10) that are used for ranking knowledge or abilities represent ordered set. Thus, mark 10 does not show that the person's knowledge is 2 times better then one's that has mark 5. Attributes of this type are also discrete values.
- CONTINUOUS: attributes with the values that form a continuous curve. Values are naturally ordered and have implicit distance and magnitude semantics. Domain for an attribute, which has this type, may also have a distribution associated with it. It gives a hint to the data mining provider about the expected distribution of the column values that will be inserted into model when it is trained. Some examples of distributions are: NORMAL, LOG_NORMAL, UNIFORM, BINOMIAL, POISSON, and so on.
- DISCRETIZED data of this type are inserted into model as CONTINUOUS, but the provider transforms and models them into specific number of ORDERED states.

When some of the possible column types (previously indicated or described in [OLEDB]) are used for creating a model, they give the provider some sense about the training data that it will be given with the INSERT command. The provider can be given some more hints that improve the quality of the built model, but they are usually provider-specific. For example:

- MODEL_EXISTENCE_ONLY indicates that the actual values for an attribute are not nearly as important as the simple existence of the attribute.
- NOT NULL indicates that the attribute can never contain a null value, and encountering one while training should generate an error.

Both, regular attributes and TABLE type columns can be input, output, or input/output column values. Attribute or table type columns can be input columns, output columns, or both. A *data mining model* should be able to predict or explain output column values based on values of the input columns. A prediction can be expressed as a histogram that provides multiple possible prediction values together with probability and other statistics.

5.2.2.2 CREATE MINING MODEL

As indicated in the previous section a data mining model can be defined by using the CREATE MINING MODEL statement. With this statement, only structure and properties of a mining model are defined, i.e. column types (refer to section 5.2.2.1) and data mining algorithm are specified. It has the following syntax:

```
CREATE MINING MODEL <mining model name>
(<Column definitions>) USING <Service>[(<service arguments>)]
```

For the student example, the `CREATE MINING MODEL` statement would have the following structure:

Query 5.3:Create Mining Model

```
CREATE MINING MODEL [StudyLengthGuess](
    [Student ID] LONG KEY,
    [group] TEXT DISCRETE,
    [semester] DOUBLE DISCRETIZED(),
    [age] DOUBLE,
    [mns] DOUBLE DISCRETIZED() PREDICT,
    [mns probability] DOUBLE PROBABILITY OF [mns],
    [exams passed] TABLE(
           [exam] TEXT KEY,
           [mark] DOUBLE DISCRETIZED(),
           [examination period] TEXT
    )
    [Courses chosen] TABLE (
           [course] TEXT KEY,
           [course probability] DOUBLE PROBABILITY OF [course],
           [type] DISCRETE RELATED TO [course]
    )
) USING [Microsoft_Decision_Trees]
```

After the *mining model* has been created it has to be populated with the training data using the `INSERT INTO` statement. During this process, the training data is run through the *data mining* algorithm and the predictive model is defined (it is also referred to as DMM content). The DMM content is the set of rules, form ulas, classifications, distributions, nodes, or any other information that was derived was derived from a specific set of data using a data mining technique. Beside that for each of the attributes, the set of all possible states is stored (if the columns are of the discrete type). For continuous attributes the minim um, average and maximum values are stored. Since it is possible that values of some attributes from the input set are not known, input set is attached additional `NULL` (*missing*) value. Depending on technique, used while creating, the type of content is differed from one model to another. The training stage includes intensive data processing, and it can last very long.

5.2.2.3 SELECT

The prediction information can be very rich, so it is often necessary to extract only a portion of the predictions. Different mining models support different requests, e.g. 'best estimate', ' top 5 estimates', … Therefore, the output column defines what information can be extracted out of it. OLE DB for Data Mining defines a set of standard functions that allow this extrac tion of information from output columns and are used in SELECT statement. Some of these functions are:

- *Predict*: Directly selecting a predictable column from a DMM is a shortcut for using the default behavior of the *Predict* function on the column. It will return the "best" predicted value for the column. Syntax for this function is Predict(`<column reference>, option1, option2,...`)(see [OLEDB]).
- *Cluster* is function, which can be used in the clustering models. A result of this function is returns a scalar value of cluster identifier that the input case belongs to with the highest probability.
- *ClusterProbability* returns the probability that the input case belongs to the cluster that has the highest probability.

The OLE DB for Data Mining `SELECT` statement has the following structure (see also Query 5.4):

```
SELECT [FLATTENED] <SELECT-expressions>
FROM <mining model name> PREDICTION JOIN <source data query>
   ON <join condition> [WHERE <WHERE-expression>]
```

The **`<SELECT-expressions>`** clause is set of comma-separated expressions, where each of them can be just a simple reference to a column from the data mining model or source data query or a general expression containing prediction functions.

I PREDICTION JOIN

Operation `PREDICTION JOIN` is used when retrieving predictions from mining model to match up actual cases from **`<source data query>`** with all possible cases from a **`<mining model name>`**. It takes one case from the input set and finds, in the mining model, matching cases using the conditions in the ON clause. (**`<join condition>`** expression). Algorithm (refer to section 3.3.1.5), then prunes this set of matching cases into one aggregate case. This case contains the best predictions for all predictable columns in the model (in the query 5.7 these are the values of the functions `Cluster()` and `ClusterProbability()`).

Though similarity in syntax to standard relational `JOIN` exists, this operation does not follow its semantics:

- Operation `PREDICTION JOIN` matches exact continuous value of the source case attributes (given by **`<source data query>`**) with some learned distribution in the mining model, and not to every possible value in predictable column. If we assume that the model *StudyLengthGuess* column *mns* has the following set of possible cases `{min, max, average, missing}` = `{8, 15, 10, NULL}` then query:

  ```
  SELECT * FROM StudyLengthGuess WHERE age=22 and mns=9
  ```

  does not return records, since this query but on `PREDICTION JOIN`, between the table containing new cases and *mining model* return learn distribution for the current attribute. For example, *group* for students 22 years old; for example *(architecture .321; computer science .276; design .225; law .113; missing .065).*

- The cases in the model represent all possible states for a column being predicted, together with the probability that this value is correct. A user selecting a prediction for a column often expects to get the single *best* predicated state. Actually, query:
`SELECT * FROM StudyLengthGuess WHERE age=22 and mns=10`
As a result returns the table:

Group	…	age	mns	mns prob
Architecture	…	22	10	0.321
Computer science	…	22	10	0.276
Design	…	22	10	0.225
Law	…	22	10	…
Missing	…	22	10	…

while `SELECT * FROM PREDICTION JOIN` returns only the most probable value (here it is *architecture*)

- The `PREDICTION JOIN` may need to make some aggregations and assumptions when confronted with missing values in the source case. If, for example, we have `SINGLETON` query:

```
SELECT * FROM StudyLengthGuess as SLG
PREDICTION JOIN (SELECT 6 as semester,
((SELECT ‘Arts’ as eExam, 10 as mark,
‘SS00’ as [examination period]) UNION …) as Exams) as case
ON SLG.semester=case.semester AND ...
```

generated as a `PREDICTION JOIN` of a model and a source case, which is not completely defined (the values for attributes `age`, `group`... are missing). As a result, just one, the most probable value for the *mns* attribute is returned, while the operation `JOIN` would return all tuples that satisfy given terms.

The **`<source data query>`** clause identifies the set of cases that will have their attributes predicted by combining them with the learned knowledge in the mining model. Depending on the provider, some or all of the following query types are supported: `SINGLETON CONSTANT`, `SINGLETON SELECT`, `OPENROWSET`, `SELECT` and `SHAPE` [OLEDB].

II ON and condition in the JOIN operation

Key columns on the case row are only for the bookkeeping and consistency reasons. Depending on the mining algorithm, key values (`KEY`) from the set of training data may not be used by the mining model. Usually, a mining model does not retain the set of distinct values for these columns. However, because each row from the models set of all possible cases is unique, it can be matched to rows from the source query of actual cases through the **`<join condition>`** clause of the ON keyword. The join condition matches the columns from mining model to columns of the source query. The join condition has one "=" expression for each set of columns to be matched and the expressions are joined with the `AND` keyword. Column references can be simple column names, they can be prefixed with a model or alias name to scope namespaces and resolve name conflicts, and they can have many scope levels to identify columns, which are in turn members of table type columns.

The `<WHERE-expression>` supports a simplified form of SQL WHERE clause semantics that can limit the cases returned from a prediction query.

5.2.2.4 SHAPE

Although, for population of many mining models, a result set with nested tables is needed, a single query to most relational providers usually cannot return it. Therefore, multiple queries have to be executed in the data source to retrieve all the data that represents the case. The queries have to be shaped into a nested table so they can be fed into the mining model. ***OLE DB for Data Mining*** provides several alternatives for performing this operation: integrated support for `SHAPE` syntax (used here), use of the ***MDAC Data Shaping Service*** (an OLE Provider that can be layered on the top of other providers and invoked in OLEDB for DM via `OPENROWSET`) or native support for nested tables (which exist in a small number of databases). In query 5.6 the basic syntax of the SHAPE statement can be recognized:

```
SHAPE {<master query>}
  APPEND({<child table query>}
  RELATE <master column> TO <child column>)
  AS <column table name>
[
  APPEND ({<child table query>}
  RELATE <master column> TO <child column>)
  AS <column table name>
]
```

It allows the addition of table columns to a master query by specifying the child table rows and the way to match between the row in the **`<master query>`** and its child rows **`<child table query>`**.

5.2.2.5 OPENROWSET

The OPENROWSET function allows the <source data query> to read the data from an external data source since most of data mining providers are not embedded in the RDBMS, which contains source data. The basic syntax of this function is:

```
OPENROWSET('provider_name','provider_string','query_syntax'),
```

where provider_**`name`** name is an OLE DB provider name (`'SQLOLEDB.1'` when using SQL Server), **`provider_string`** is the OLE DB connection string for that provider.

In case of just reading data to shape them into nested table (but not inserting them into mining model) the connection string has the following format:

```
‘Persist Security Info = True; User ID = userid; Password = pass-
word; Initial Catalog = Sweets; Data Source= dbServer’,
```

while in case of populating data mining model (when using `INSERT` statement):

```
'Provider=SQLOLEDB.1; Integrated Security=SSPI; Persist Security
Info=False; InitialCatalog = Sweets; Data Source= dbServer'
```

Parameter **`query_syntax`** is a query that returns a row set. The data mining provider will establish the connection to the data source using the **`provider_name`** and **`provider_string`** and will execute the query specified in **`query_syntax,`** to retrieve the source data rowset.

5.2.2.6 DELETE, DROP

There are two possible ways to delete existing mining model. First one is to delete the data mining model object i.e. remove the object from the system, with both its structure and its content. For this operation statement `DROP MINING MODEL <model name>` is used. The other operation clears just the content of the data mining model object but leaves its structure intact. For this operation `DELETE` statement is used. The `DELETE` statement can be used in two variations:

- `DELETE FROM <model name>` deletes the content and the column values but the structure stays. Then the data mining model can be repopulated with a new set of training data.
- `DELETE FROM <model name>.CONTENT` deletes the content of the mining model, but leaves the structure and the learned values. This means that the learned patterns are dropped but that the predicted values stay.

5.3 Advisor tag library

This is a library of tags that implement clustering functionality. The recommendation module is implemented as a sub-package of the `portal.taglib` package. The basic idea for using clustering has already been suggested in 5.1. In this part, the implementation will be explained in few more details.

The recommendation module forms clusters of the users according to their personalities. The *Personality* of a user is the whole of the selections he made from a global information space and the weight and ordering he is attributing to it. During his work and interaction with other people, personality defines what the single user knows for him, what he gives to others and how he integrates information from other people. A person, usually, participates in many different communities and in each of them bears a specific role. His *Role* in a single community is the part of his personality i.e. subset of all the choices that user made. The role is used for getting recommendations from a specific community. This actually means that the user neighborhood discovery is performed in a specific context, i.e. the set of applications, used by the particular community. Groups, usually, have sense only in the particular application context, in which they are generated, and can very rarely be used in some other.

The consequence is that the user profile structure, in this context (i.e. in the domain of the specific role) is flat. Hence, in the clustering process all dimensions have the same influence. Knowing these limitations the utilization of ***advisor*** tag library becomes very simple. Extensions that would include more attributes or different mining models are straightforward and easy to implement, but not so easy to use since it requires much broader programming knowledge and extensive explanation (refer to section 3.5).

Figure 5.5 shows the extended portal framework (refer to section 4.1). Tag library ***advisor*** is a part of the package `portal.taglib`, and includes two smaller packages `advisor.clustering` and `advisor.ado`.

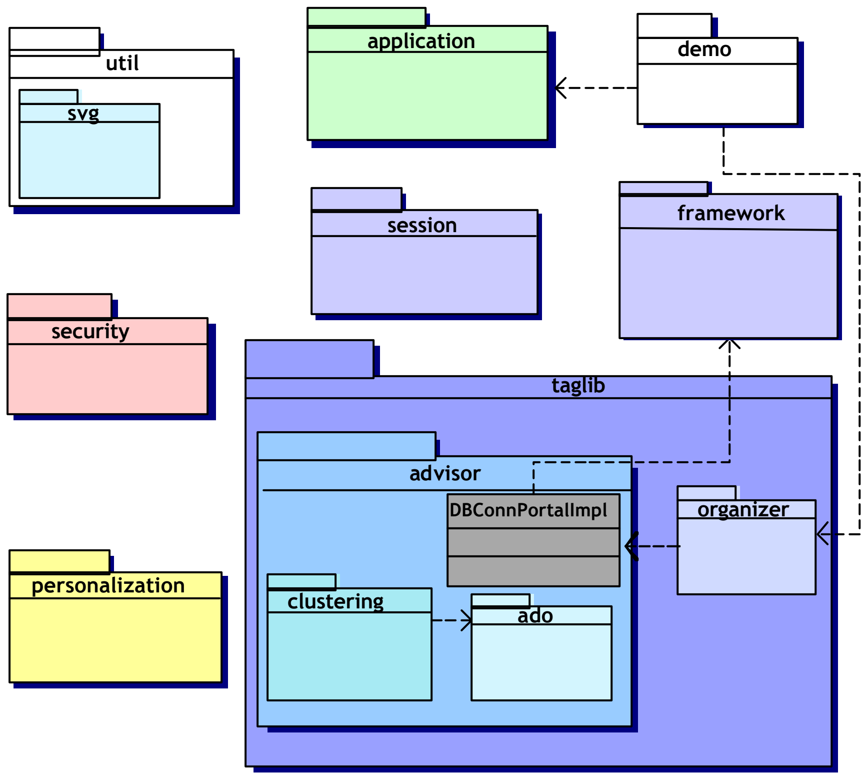


Figure 5.5 Extending portal framework with ***advisor*** module

The `advisor.clustering` package contains tags, which allow creation, manipulation, and destruction of a clustering model.

The `advisor.ado` package contains five classes, and one interface that provide direct access to data stored by a data mining provider. The classes from this package, actually, wrap ***ActiveX Data Objects*** (ADO) [ADO] for JACOB, and provide access to various kinds of data sources through an OLE DB provider. If there is no OLE DB provider for a specific database, the connection can be made through the OLE DB provider for ODBC Drivers. This provider effectively grants access to any data source that has ODBC driver i.e. nearly all commonly used databases. All of the classes are derived from `com.jacob.com.Dispatcher` class (refer to section 5.2.1), as it will be explained later in more detail (refer to section 5.3.2). Interface `CommandTypeEnum` stores values of ADO options used to optimize how commands are executed (see appendix D).

The `DBConnPortalImpl` class is the only class dependant on the portal framework. It takes a connection from the portal context (action 1.5.1.2 in figure 5.6), which then undertakes it from the database connection pool (action 1.5.1.2.1); it wraps creation of a statement (action 1.5.2), which, is done by the connection taken from the pool (action 1.5.2.1.1) returns the statement to the tag that has acquired it. After the query execution it returns the connection to the pool (action 1.5.8.1.2- `releaseConnection`).

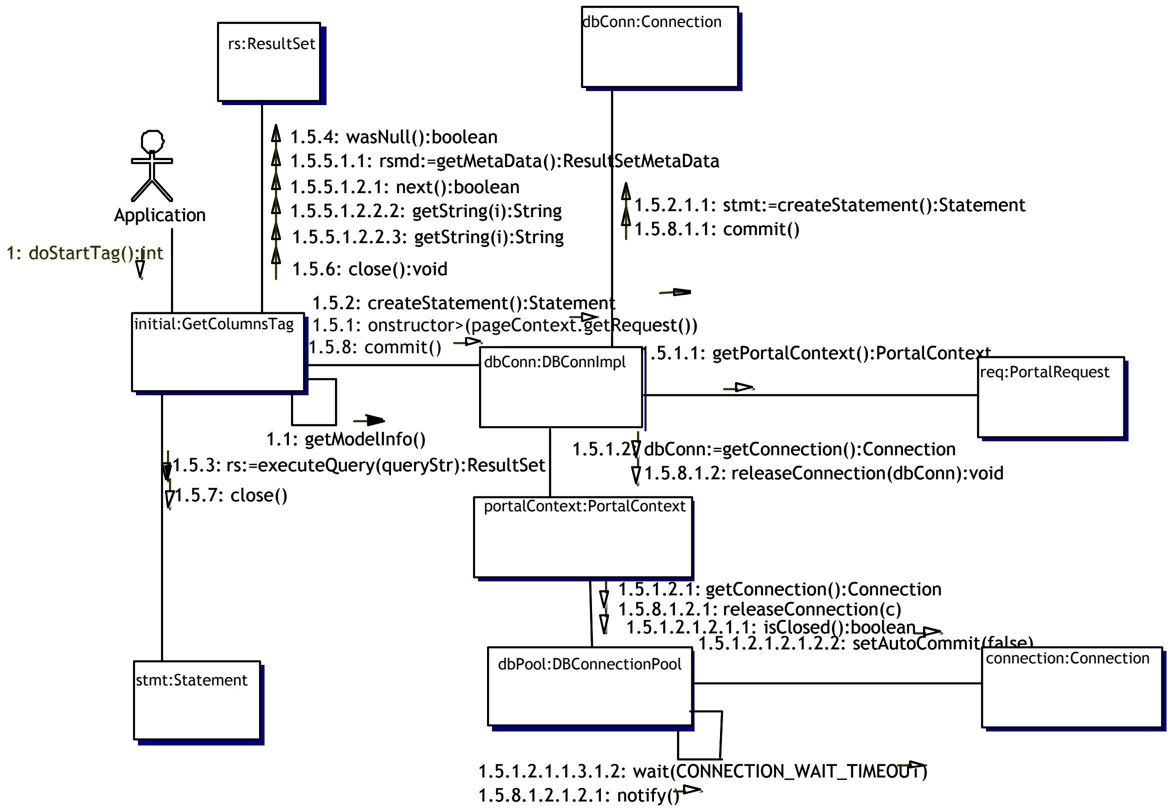

rs:ResultSet
dbConn:Connection
Application
1: doStartTag():int
1.5.4: wasNull():boolean
1.5.5.1.1: rsmd:=getMetaData():ResultSetMetaData
1.5.5.1.2.1: next():boolean
1.5.5.1.2.2.2: getString(i):String
1.5.5.1.2.2.3: getString(i):String
1.5.6: close():void
1.5.2.1.1: stmt:=createStatement():Statement
1.5.8.1.1: commit()
1.5.2: createStatement():Statement
1.5.1: onstructor>(pageContext.getRequest())
1.5.8: commit()
initial:GetColumnsTag
dbConn:DBConnImpl
1.5.1.1: getPortalContext():PortalContext
req:PortalRequest
1.1: getModelInfo()
1.5.3: rs:=executeQuery(queryStr):ResultSet
1.5.7: close()
1.5.1.2: dbConn:=getConnection():Connection
1.5.8.1.2: releaseConnection(dbConn):void
portalContext:PortalContext
1.5.1.2.1: getConnection():Connection
1.5.8.1.2.1: releaseConnection(c)
1.5.1.2.1.2.1.1: isClosed():boolean
1.5.1.2.1.2.1.2.2: setAutoCommit(false)
stmt:Statement
dbPool:DBConnectionPool
connection:Connection
1.5.1.2.1.1.3.1.2: wait(CONNECTION_WAIT_TIMEOUT)
1.5.8.1.2.1.2.1: notify()

Figure 5.6 Collaboration diagram for DBConnPortalImpl

In the following few sections the main concepts of the ***advisor*** tag library will be explained by taking closer look at some of the tags.

5.3.1 advisor.Clustering

The `advisor.clustering` package contains tags, which allow creation, manipulation, and destruction of a clustering model, as well as only its content. Tags contained in the library are shown in figure 5.7. Using these tags it is possible to:

- create new user clusters simply by changing the criteria according to which they are clustered;
- Retrain existing data for new clusters,
- Find the cluster a specific user belongs to and all information about it,
- Drop all the data about the specified mining model

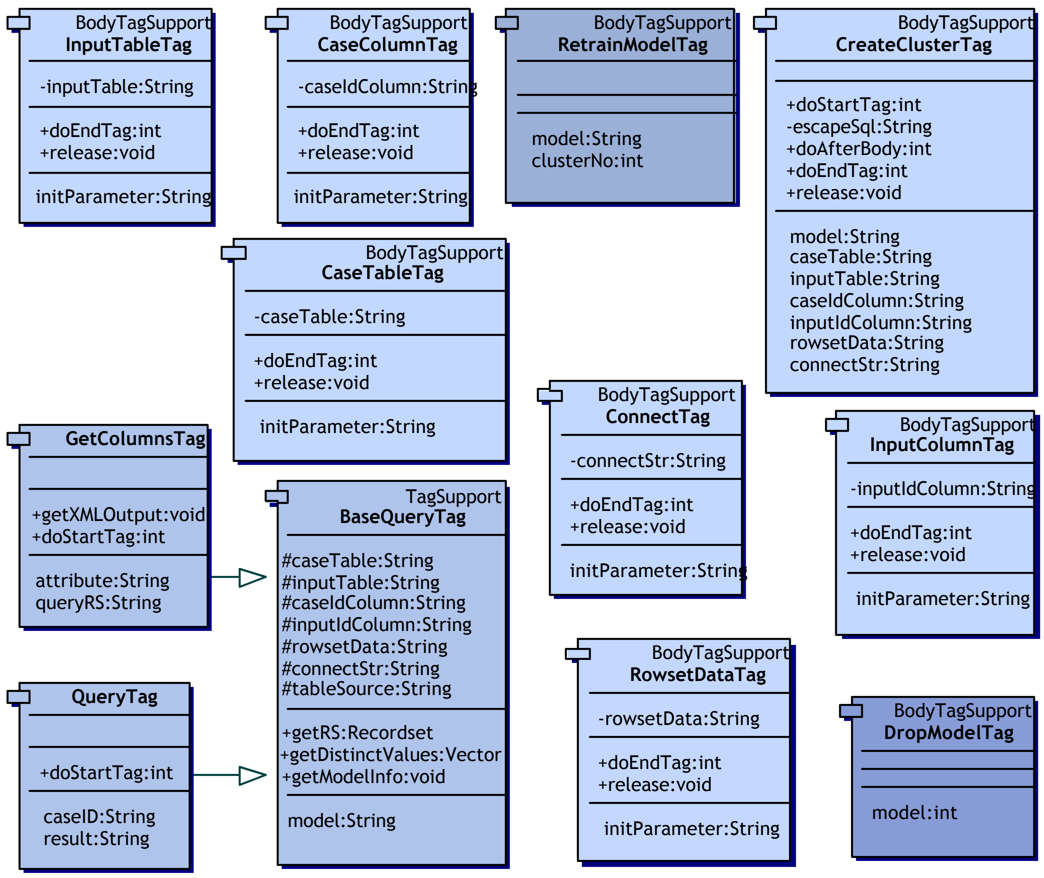


Figure 5.7 Class diagram for the `advisor.clustering` package

5.3.1.1 *CreateClustersTag*

This tag allows the creation of the clustering mining model. Since the model is created just once, and then different queries can be performed or retraining, this tag can be used only once (form one model) on the administrator page. Its utilization is not required. However, if the `<createClusters/>` tag is not used, a mining model has to be created some other way e.g. by using a data mining modeling tool (like ***Microsoft Analysis Services Front-End***). The model, created with this tag and used for recommendation has to fit the following constraints:

- Key attribute (`caseColumn`) of the model contains `caseIDs`, which represent unique identifiers for entities that are clustered. Basic description of the entities (i.e. cases) can be found in `caseTable`.
- The cases are clustered based on the information stored in `inputTable`. In the relational database, `inputTable` represents the table containing multi-

valued attributes for the cases; actually, for each tuple in that table the combination (`caseColumn`, `inputColumn`) represents key attribute.

Microsoft OLE DB for Data Mining query used for model creation has the format as indicated in the query 5.4, where the values for the parameters (italic text) have to be provided by the application programmer through suitable tag utilization. For example, the sequence from an xJSP application, indicated by example 5.5, is used for creation of the clustering model *RecipeBook that* clusters all *cakes* based on their *ingredients*.

Query 5.4:Create Clustering Model

```
CREATE MINING MODEL [model]
   ([caseColumn] TEXT KEY,
   [inputTable] TABLE
   ([inputColumn] TEXT KEY),
   [caseColumn1] TEXT DISCRETE PREDICT_ONLY)
   USING Microsoft_Clustering (CLUSTER_COUNT = noOfclusters);
```

It is necessary that the tables: *cakes*, with primary key *cakeName* (where all the information on cakes is stored) and *ingredients,* with primary key *[cakeName, ingredient]* (where all the information about ingredients of indicated cakes is stored) exist in the database *Sweets* on *dbServer*. The necessary information for model creation is set by setter methods, which are tags themselves, nested in the `create` tag. For that reason, the `CreateClustersTag` class has to manipulate its body; thus, it extends `BodyTagSupport` (refer to section 4.4.2.2). All the parameters are passed to the parent tag (`create`) through the JSP `pageContext` as attributes and therefore available to it before the method `doEndTag()` starts its execution.

Figure 5.8 shows the sequence of actions executed during model creation (execution of `doEndTag()` method). After opening the connection to the COM object (notice that the parameter `adCmdUnspecified` is compulsory if the query returns an empty `RecordSet`), a sequence of commands is executed first to create the mining model and then to train it with existing data (actions 1.2.8 and 1.2.10 on the figure 5.8). The occupied COM object must be released after all the actions are executed (action1.4).

After the creation, a description of the model is stored in the database system table `AdvisorModels` and can be retrieved by executing the `<getModelInfo/>`.tag.

5.3.1.2 QUERYTAG

This tag queries an already existing and trained mining *model.* The result of the execution are all entries from the *caseColumn,* which are members of the *cluster* the given *caseID* belongs to. The parameter *caseID* has to be one of the entries in *caseColumn,* which exist both in *caseTable* and *inputTable* tables in the source database. Otherwise, the returned *result* will be empty.

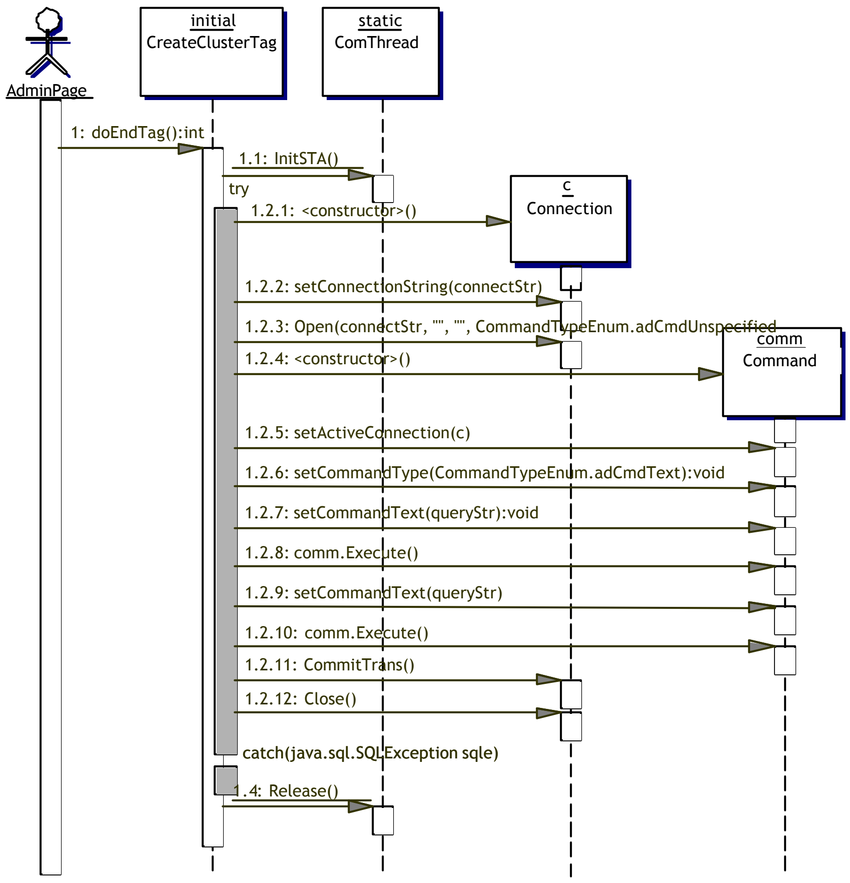

AdminPage
initial CreateClusterTag
static ComThread
c Connection
comm Command
1: doEndTag():int
1.1: InitSTA()
try
1.2.1: <constructor>()
1.2.2: setConnectionString(connectStr)
1.2.3: Open(connectStr, "", "", CommandTypeEnum.adCmdUnspecified
1.2.4: <constructor>()
1.2.5: setActiveConnection(c)
1.2.6: setCommandType(CommandTypeEnum.adCmdText):void
1.2.7: setCommandText(queryStr):void
1.2.8: comm.Execute()
1.2.9: setCommandText(queryStr)
1.2.10: comm.Execute()
1.2.11: CommitTrans()
1.2.12: Close()
catch(java.sql.SQLException sqle)
1.4: Release()

Figure 5.8 `CreateClustersTag:doEndTag()`sequence diagram

Example 5.5:Creation of the Clustering Model using tags

```
<harry:create model="RecipeBook">
   <harry:caseTable>cakes</harry:caseTable>
   <harry:inputTable>ingredients</harry:inputTable>
   <harry:caseColumn>cakeName</harry:caseColumn>
   <harry:inputColumn>ingredient</harry:inputColumn>
   <harry:connect>Location=dbServer;Provider=msolap;MiningLocation=c:/myr
   ecipes;</harry:connect>
   <harry:rowsetData>'SQLOLEDB.1','Persist Security Info = True; User
   ID=userid; Password=password; Initial Catalog = Sweets; Data Source=
   dbServer'</harry:rowsetData>
   <harry:sourceTable>'SQLOLEDB.1','Provider=SQLOLEDB.1;Integrated Secu-
   rity=SSPI; Persist Security Info=False; InitialCatalog = Sweets; Data
   Source= dbServer'</harry:sourceTable>
</harry:create>
```

The *recipe book* example shows that this makes sense, because it is not possible to say whether *myCake* falls in the banana cake or *Schwarzwalder* cake category if none of its ingredients is already known. In the example 5.6 one application of this tag is shown. Model *RecipeBook* is queried for names of all the cakes that have the same flavor (i.e. ingredients as *myCake*).

The result is a vector of all names of the cakes that have the same flavor (i.e. belong to the same cluster) as *myCake,* for example *SameFlavo rCakes={'myCake', 'Schwarzwalder cake', 'Reform',...}.*

Example 5.6:An application of the query tag

```
….
<advisor:query model="RecipeBook" caseID="<%= myCake %>" re-
sult="SameFlavorCakes" />
….
```

The assumption during the execution of this tag is that it can get the metadata about the mining model it should use. This is achieved through the `pageContext` default variable. Thus, in the xJSP application, before this tag is referenced, there has to exist a reference to `<getModelInfo/>`, which will put the required data in `pageContext`. The `QueryTag` class inherits the `BaseQueryTag class` (figure 5.7), and its methods `getModelInfo()`, `getRS()` and `getDistinctValues()`. These methods create the query to the DMM stored by the *data mining* provider (like ***Microsoft Analysis Services***), using the indicated parameters; and return the requested data to the application. This is achieved by employing the wrapper classes for ADO objects (`Recordset`, `Field`…), and are also the parts of the `ad-visor` package (refer to section 5.3.2).

Query 5.7: query Mining Model

```
SELECT FLATTENED
 [T1].inputTable, Cluster() AS Node, ClusterProbability() AS CProb
FROM model PREDICTION JOIN
    SHAPE {OPENROWSET (rowsetData,
    ’SELECT caseColumn AS caseColumn
    FROM caseTable
    ORDER BY caseColumn’)
    }APPEND(
    {OPENROWSET (rowsetData,
    ’SELECT caseColumn AS caseColumn_1,
    inputColumn AS inputColumn
    FROM inputTable
    WHERE caseColumn =’caseID’
    ORDER BY caseColumn)}
    RELATE caseColumn TO caseColumn_1)
    AS inputTable
    AS [T1]
ON
 model.caseColumn = [T1].caseColumn AND
 model.inputTable.inputColumn = [T1].inputTable.inputColumn;
```

Method `getRS(connectionStr, queryStr)` returns the *Recordset* object that contains the results of a query posed to the *model*. A prediction query is embedded in `QueryTag`, and specified by `queryStr` (query 5.7). In this case, everything has already been specified when the mining model was created, so only one parameter can be customized (*caseID*).

For the *recipe book* example, the `SHAPE` statement from the query 5.7 has the following format:

```
SHAPE {
        OPENROWSET (rowsetData,
        'SELECT cakeName AS cakeName
        FROM cakes
        ORDER BY cakeName)
}APPEND(
{
        OPENROWSET (rowsetData,
        'SELECT cakeName AS cakeName_1,
        ingredient AS ingredient
        FROM ingredients
        WHERE cakeName ='myCake'
        ORDER BY cakeName)}
RELATE cakeName TO cakeName _1)
AS ingredients
```

where the source tables are:

Cakes	
cakeName	...
myCake	...
Reform	...
...	...

ingredients		
cakeName	ingredient	quantity
myCake	sugar	...
Reform	eggs	...
...	...	...

And the resulting table containing complete cases is:

CakeName	ingredients	
myCake	cakeName_1	ingredient
	myCake	eggs
	myCake	flour
	…	…

5.3.2 advisor.ado

The classes from the `advisor.ado` package provide direct access to data stored in any, database alike, data source; in this case it is a data mining provider. The classes are derived from the main JACOB class `com.jacob.com.Dispatch` (refer to section 5.2.1) as wrappers for *ADO*. They are source-code equivalent to those produced by the *Microsoft JACTIVEX* tool. Interface `CommandTypeEnum` stores values of ADO options used to optimize how commands are executed (see appendix D).

5.3.2.1 RECORDSET

The `Recordset` object is the COM counterpart of the ODBC resultset dressed up with some extremely useful facilities, such as bookmarking, filtering, and sorting. Since it is a COM object it can hardly be exchanged with modules running on other non-Windows platforms. This class, together with all the other ADO wrappers for JACOB automation classes, allows direct access to the data stored in a database-like storage. It extends class `com.jacob.com.Dispatch` and delegates all the methods to the underlying `IDispatch` pointer.

- The `Recordset` class constructor calls `com.jacob.com.Dispatch` constructor, which takes an Registry `ProgID` as an argument and creates new instance of the component specified by that identifier (in the following example that is string value *ADODB.Recordset*):

```
public Recordset(){
  super("ADODB.Recordset");
}
```

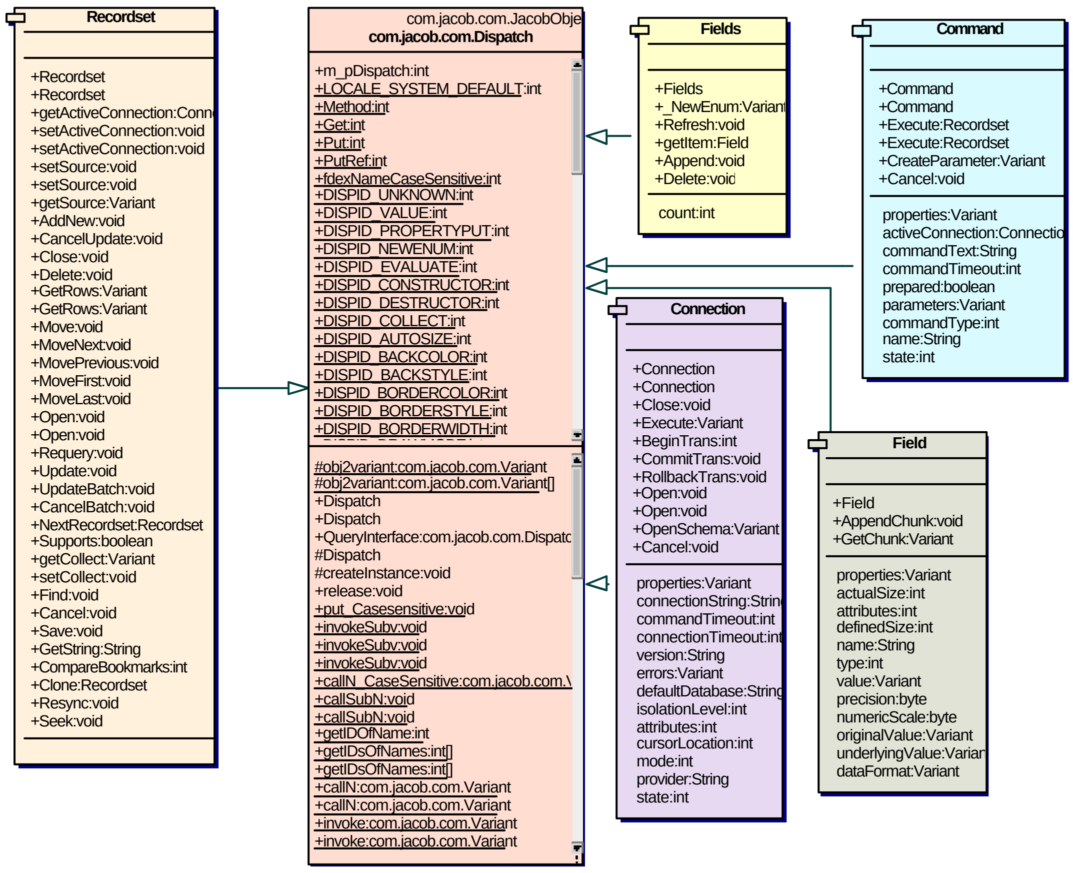


Figure 5.9 ADO wrapper classes

In order to return ADO *wrapper* as strongly typed wrapper from method call, a special constructor has to be created. This constructor makes the wrapper class instance and copies over the `IDispatch` reference. This is because a java `Dispatch` object can't be cast to a subclass object. For example, the `Recordset` class has a method `public Fields getFields()from the Recordset` class.

Ideally, wrapper code should wrap the returned `Dispatch` reference in a `Fields` object:

```
public Fields getFields(){
     return new Fields(Dispatch.get(this, "Fields").toDispatch());
}
```

- Method `getFields()` uses a special constructor inserted into the `Fields` class. This constructor is used instead of a case operation to turn a `Dispatch` object into an ADO wrapper object - it must exist in every wrapper class whose instances may be returned from method calls wrapped in `VT_DISPATCH` Variants.

```
public Fields(Dispatch d){
  // gets IDispatch pointer
  m_pDispatch = d.m_pDispatch;
  // sets the pointer value on null
  d.m_pDispatch = 0;
}
```

The remainder methods from the `Recordset` class wraps calls to methods from the `Dispatch` class, for example:

- Method `Open (Variant source, Variant activeConnection, int cursorType, int lockType, int options)`, together with all of it's variations with less parameters, opens the cursor. Parameter `source` is a `Variant` object that can be cast to a valid `Command` object, SQL statement, table name, call to the stored procedure, `URI` or name of `File` or `Stream` object, that contains a `Recordset`. The `activeConnection` parameter is a `Variant` that can cast into a valid `Connection` object or a `connection` String. The `cursorType` parameter is `AdoEnums.CursorType` value, which defines the type of the cursor, used by the provider when opening a `Recordset` (default value is `FORWARDONLY` – see appendix D). Parameter `lockType` contains value `AdoEnums.LockType` that indicates which type of locking (*concurrency control*) should be used by provider when `Recordset` is opened (default value is `READONLY`). The `options` parameter is integer value that informs the provider which way to evaluate argument `source`, if it is not `Command` object; or what to do when the `Recordset` should be loaded from the file it is stored in (it can contain one or more `AdoEnums.CommandType` or `AdoEnums.ExecuteOption` values). This method wraps the call:
```
Dispatch.call(this, "Open", Source, ActiveConnection,
          new Variant(CursorType), new Variant(LockType),
          new Variant(Options))
```
- Methods `getEOF()`, `MoveFirst()`, `MoveNext()` enable moving through the `Recordset`. Method `getEOF()` indicates that the active record pointer is set behind the last record in the `Recordset` object. Methods `MoveFirst()` and `MoveNext()` move the pointer on active record on the first, and the next record in the active `Recordset` object, respectively.

5.4 Organizer tag library

This is a utility tag library consisting of tags that are initially developed for specific application purposes. However, most of the tags implement functionalities that can be reused; for example, output a vector in XML format or filter elements of a vector according to some condition.

Figure 5.10 shows the class diagram of the package `portal.taglib.organizer`. It contains a few helper classes and tags that are specific for the *Interactive timetable* application (`EventListTag, LectureEvents, FreeEvents`), which will be explained later (refer to section 5.5). In this section the generic classes `CustomEventListTag`, `FilterTag` and `OutputXMLTag` will be described

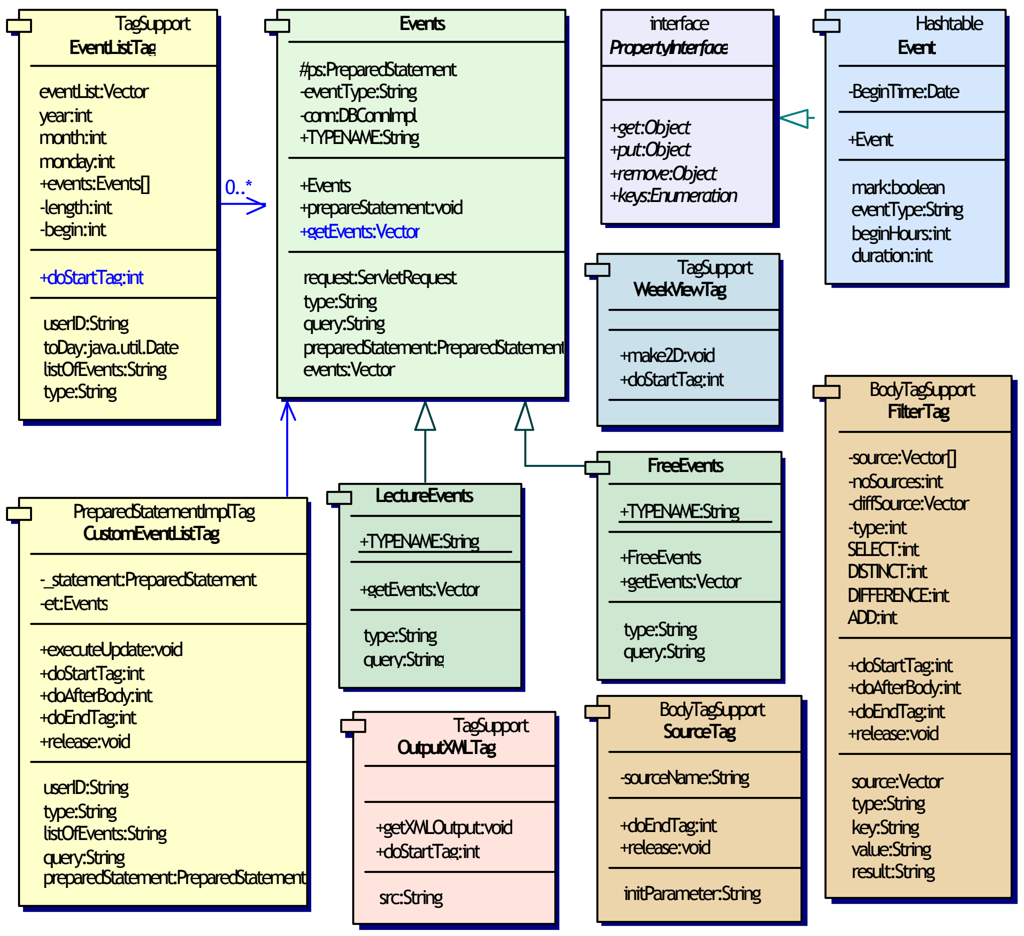


Figure 5.10 Class diagram for the `Organizer` tag library

5.4.1 CustomEventListTag

This tag, for a given `userID` and specified database `query`, returns the `listOfEvents`. The `listOfEvents` is a vector of *Event*s or properties of some, specified *type*. These objects are retrieved as a `ResultSet` of specified query, where each of them wraps one row. It is essential that elements of the vector are instances of classes that implement `PropertyInterface`. Initially, this vector is stored as an attribute in `PAGE_CONTEXT` under the name `listOfEvents` (it can be changed by setting the `scope` parameter).

The `CustomEventListTag` class extends `dbtags.PreparedStatementImplTag` from `org.apache.taglibs.dbtags` tag library (figure 5.11), and therefore inherits its nested tags `<query/>`, `<execute/>` and `<setColumn/>`. Alas, there are few differences between these two tags:

- Unlike the `PreparedStatementImplTag` class, `CustomEventListTag` wraps all database dependant operations. This means that a JSP programmer does not have to open the connection to the database in the JSP application. When using the ***ETH World*** portal framework, the connection is taken from the existing database pool. For any other framework, `DBConnImpl` class has to be subclassed. On the other hand, this is also a disadvantage, since the application, using this tag, is limited to only one database.

- The result of `CustomEventListTag` execution is a vector of objects, while the result of the `preparedStatement` tag execution is an instance of the `ResultSet` class.

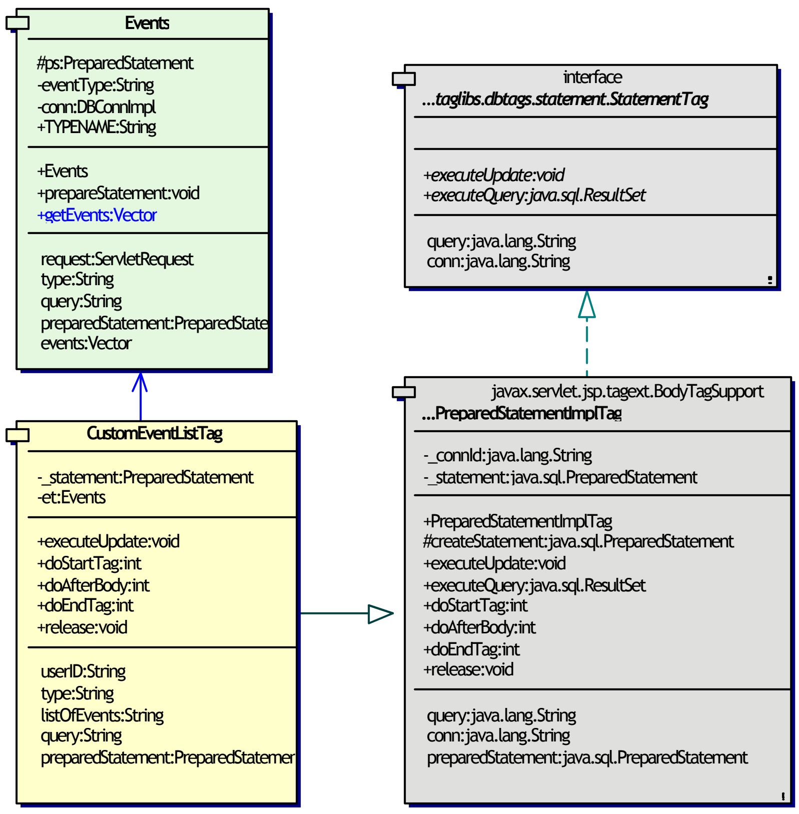


Figure 5.11 Class hierarchy for `CustomEventListTag`

Example 5.8 shows one part of the Java Server Page that uses this tag:

Example 5.8:Utilization of the CustomEventListTag tag

```
<hermione:customEventList id="ud" userID="<%= userID %>"
type="UserData" listOfEvents="uData" >
   <sql:query>
     SELECT * FROM Student WHERE StudentID = ?
   </sql:query>
   <sql:execute>
     <sql:setColumn position="1"><%= userID %></sql:setColumn>
   </sql:execute>
</hermione:customEventList>
<hermione:outputXML src="uData"/>
```

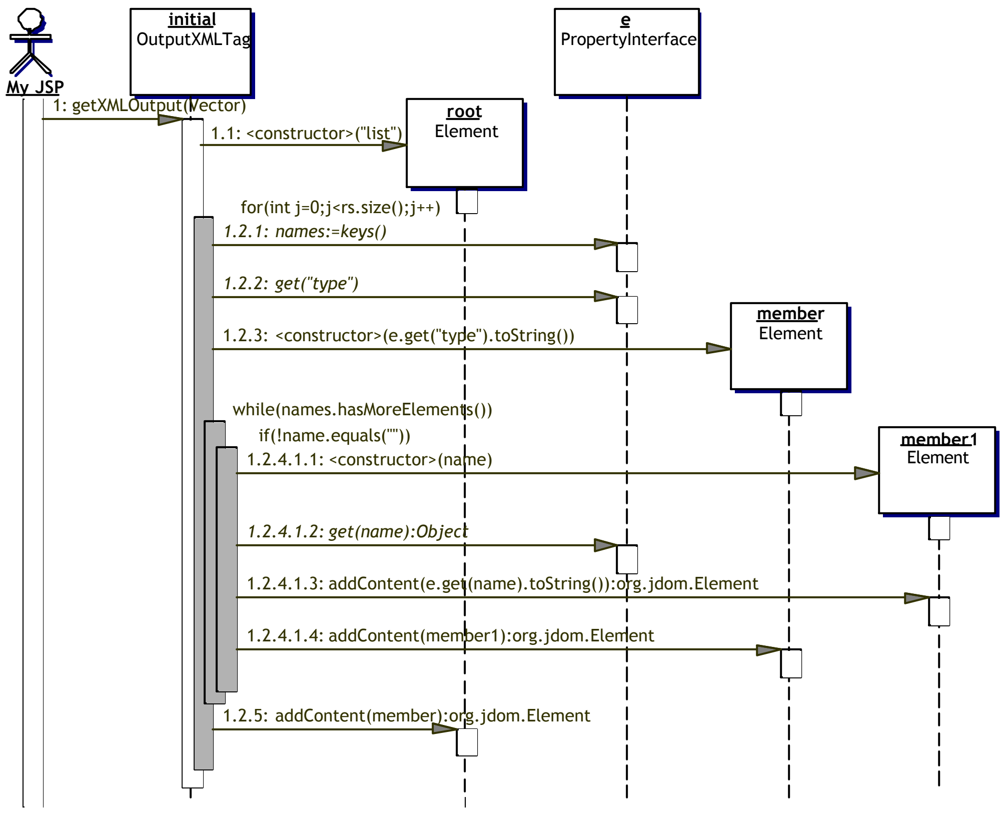
After execution of these lines, the resulting vector is outputted in XML format, as it is shown in result 5.9.

Result 5.9: CustomEventListTag output

```
<list>
   <UserData>
    <StudentID>ANCIKA</StudentID>
    ...
    <currentSemester>5</currentSemester>
    <type>UserData</type>
    <InroleYear>1998</InroleYear>
   </UserData>
</list>
```

5.4.2 OutputXMLTag

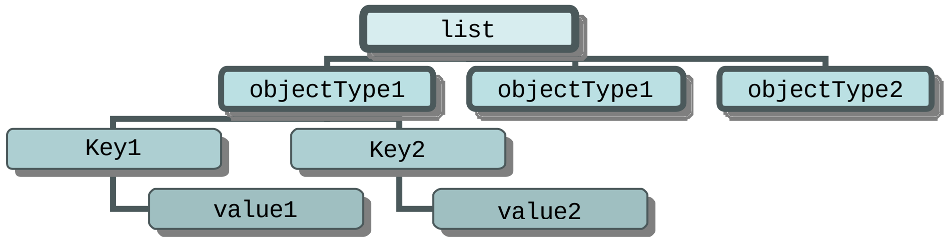
This tag outputs the vector referenced by the `src` parameter in XML format. The assumption is that all objects in the vector are instances of the classes that implement `PropertyInterface` and therefore its method `get(key)`. For the implementation of this tag, the JDOM library is used. It allows intuitive creation, manipulation, and output of XML trees.



My JSP
initial
OutputXMLTag
e
PropertyInterface
root
Element
member
Element
member1
Element
1: getXMLOutput(Vector)
1.1: <constructor>("list")
for(int j=0;j<rs.size();j++)
1.2.1: names:=keys()
1.2.2: get("type")
1.2.3: <constructor>(e.get("type").toString())
while(names.hasMoreElements())
if(!name.equals(""))
1.2.4.1.1: <constructor>(name)
1.2.4.1.2: get(name):Object
1.2.4.1.3: addContent(e.get(name).toString()):org.jdom.Element
1.2.4.1.4: addContent(member1):org.jdom.Element
1.2.5: addContent(member):org.jdom.Element

Figure 5.12 The `getXMLOutput()` method sequence diagram

As indicated on figure 5.12, creation of the XML tree is performed calling the `getXMLOutput()` method from the class `OutputXMLTag`; it is called from the `doEndTag()` method, after the source vector, specified by the `src` parameter in the `pageContext`, has been retrieved. Then the iteration through the source vector and building a JDOM `DocumentFragment` starts. For each element in the vector, the corresponding JDOM `Element` is generated (sequence 1.2.3 in the sequence diagram 5.12), whose elements are indicated by `keys` (sequences 1.2.1, 1.2.4.1.1) and content contained in values (sequence 1.2.4.1.3) of the vector object. The output tree has the following structure:



If the value of a particular attribute is `null`, then the output tree contains either default value or the `(key,null)` is not forwarded to output.

Using `XMLOutputter` class the resulting JDOM tree is formatted into a stream as XML and printed out to `JSPWriter`, so it could be submitted to the portal framework.

5.4.3 FilterTag

This tag filters out elements from a source vector, specified by the `<source/>` tag, according to a given condition. Tag `<source/>` can accept two types of input, Java `Vector` object or a XML tree. Attribute `id` holds a reference to a source vector (Java `Vector` object) in the `pageContext`, if the `<source/>` tag body does not exist. Here it is, also, assumed that all the objects in the vector instantiate class that implements `PropertyInterface`. If the `<source/>` tag body exists, then it has to be a correct XML tree; this tree is, while loading, transformed into Java `Vector` object with the name specified by `id` attribute and stored in the `pageContext`, where it waits for further processing.

It allows five different types of filtering: `SELECT`, `EXCLUDE`, `DISTINCT`, `DIFFERENCE` and `ADD`. Depending on the type indicated, it can contain different number of nested `<source/>` tags. When the filter type is `SELECT`, `EXCLUDE`, or `DISTINCT` only one source vector can be processed. In other cases, the number of source vectors is not limited. Parameter `key` indicates the condition of filtering. In case of `SELECTION`, it filters out all the elements whose attributes, which are named by `key` parameter, are not equal to the indicated `value`. For example, in the example 5.10 by applying the `filter` tag, of the `SELECT` type, on the `source` vector `recipeBook` output vector `creamy`, containing all the cakes that have as it's ingredient 1l milk, is generated.

Example 5.10: Filter tag option SELECT

```
<organizer:filter type="SELECT" key="milk" value="1l" result="creamy">
   <organizer:src id="recipeBook"/>
<organizer:filter>
```

Filter of the `EXCLUDE` type holds the function inverse to the `SELECT` filter type. When applying this type of filter on the input vector, all the elements, holding the value of the `key` attribute equivalent to indicated, are filtered out. The following relation holds:

$$filter_{select}(source, key, value) + filter_{exclude}(source, key, value) = source$$

When the type of filter is `DISTINCT` the filtering of keys is performed; actually, all distinct keys in the vector are identified. Application of this, and the rest of the filters, is completely intuitive. In that manner example 5.11 can be read as: find all cakes (vector elements) in the `RecipeBook`, which contain different amounts of milk (different values for the `milk` parameter) among their ingredients and put them into `milky` category:

Example 5.11: Filter tag option DISTINCT

```
<organizer:filter type="DISTINCT" key="milk" result="milky">
   <organizer:src="recipeBook"/>
<organizer:filter>
```

If the filter type is `DIFFERENCE`, all the elements from the first source, which have the value for attribute indicated by `key` parameter equal to some element contained in any of the other sources are filtered out. This is actually an operation difference applied in the sets of keys of the first source and all the others. If $keys_{src[i]}$ represents the set of keys for i-th `source` vector, then operation `DIFFERENCE` on n source vectors can be defined with the following expression in the set algebra:

$$result = \{...\{keys_{src[0]} \setminus keys_{src[1]}\} \setminus keys_{src[2]}\} \setminus ... \setminus keys_{src[n]}$$

In the example 5.12, from the `MyRecipeBook` vector all the elements (cakes), holding the same names as the cakes in the `Mom'sRecipeBook` vector are filtered out and new vector named `mine` is created.

Example 5.12: Filter tag option DIFFERENCE

```
<organizer:filter type="DIFFERENCE" key="name" result="mine">
   <organizer:src="MyRecipeBook" />
   <organizer:src="Mam'sRecipeBook" />
<organizer:filter>
```

The last case, `ADD` filter, does not the take parameter `key` into account. It simply all the elements of other `sources` appends to the first `<source/>`. The resulting vector of the example 5.13 is a collection of all the recipes that exist in my (`MyRecipeBook`), grannies (`GranniesRecipeBook`) and somebody-else's (`SmbdsRecipeBook`) recipe book, simply listed one after another.

Example 5.13: Filter tag option ADD

```
<organizer:filter type="ADD" result="collection">
   <organizer:src="MyRecipeBook"/>
   <organizer:src="GranniesRecipeBook"/>
   <organizer:src="SmbdsRecipeBook"/>
<organizer:filter>
```

5.4.4 PropertyInterface, Event and Events

All previously described tags work with object vectors. It is assumed that all elements of the vectors implement interface `PropertyInterface` and therefore its methods `get`, `put`, `remove`, and `keys`. This, in fact, means that all the objects of the vector are perceived as objects that have properties defined by (`key`, `value`) i.e. `(propertyName, propertyValue)` pairs and therefore can be accessed using the mentioned methods.

For the application ***Interactive Timetable*** (refer to section 5.5) the class *Event* that implements `PropertyInterface` has been developed. Along with the already mentioned methods it contains methods that allow setting and getting an event type, getting a begin time and duration of an event and marking the processed events.

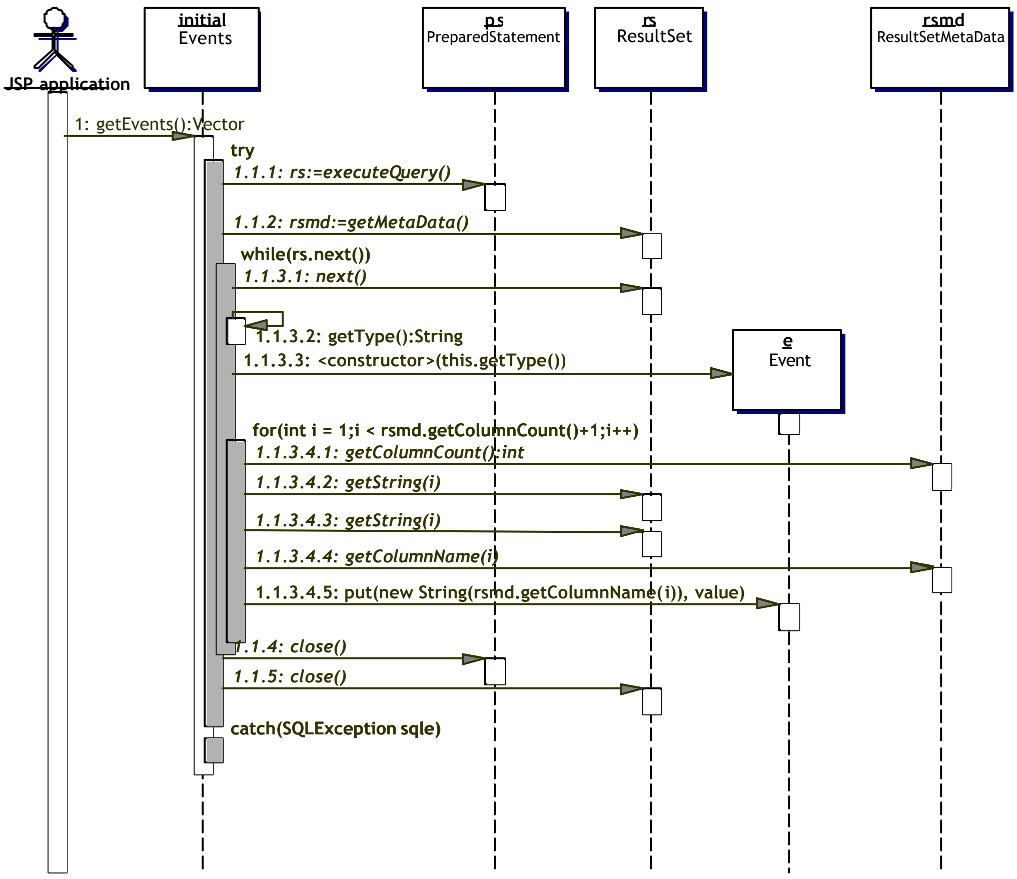

JSP application
initial Events
ps PreparedStatement
rs ResultSet
rsmd ResultSetMetaData
e Event
1: getEvents():Vector
try
1.1.1: rs:=executeQuery()
1.1.2: rsmd:=getMetaData()
while(rs.next())
1.1.3.1: next()
1.1.3.2: getType():String
1.1.3.3: <constructor>(this.getType())
for(int i = 1;i < rsmd.getColumnCount()+1;i++)
1.1.3.4.1: getColumnCount():int
1.1.3.4.2: getString(i)
1.1.3.4.3: getString(i)
1.1.3.4.4: getColumnName(i)
1.1.3.4.5: put(new String(rsmd.getColumnName(i)), value)
1.1.4: close()
1.1.5: close()
catch(SQLException sqle)

Figure 5.13 The `Events:getEvents()` method sequence diagram

The `Events` class allows generation of the list of `Event` objects from the specified database query. Each tuple from the database `ResultSet` is mapped to one `Event` object. This means that each column name is mapped to the corresponding property name and the same goes with values. A List (`Vector`) can contain various types of `Event` objects. Figure 5.13 shows the sequence of actions that are taken during the execution of the `getEvents` method.

5.5 Interactive Timetable

In the previous sections, the ***advisor*** and ***organizer*** tag libraries are explained and in this section one possible application of these two will be shown. Since the motivation for developing both, the portal framework and the recommendation library, lies in the ***ETH World*** project, ***Interactive Timetable*** as one their employment is a straightforward application of the same idea. As already mentioned, the idea of the ***ETH World*** project is to build a virtual bridge between two existing physical locations at ETH, and that way provide better interaction between ETH and its members. Since the topic of this thesis includes portals and personalization it is easy to guess that it suggests a portal as the tool for that interaction.

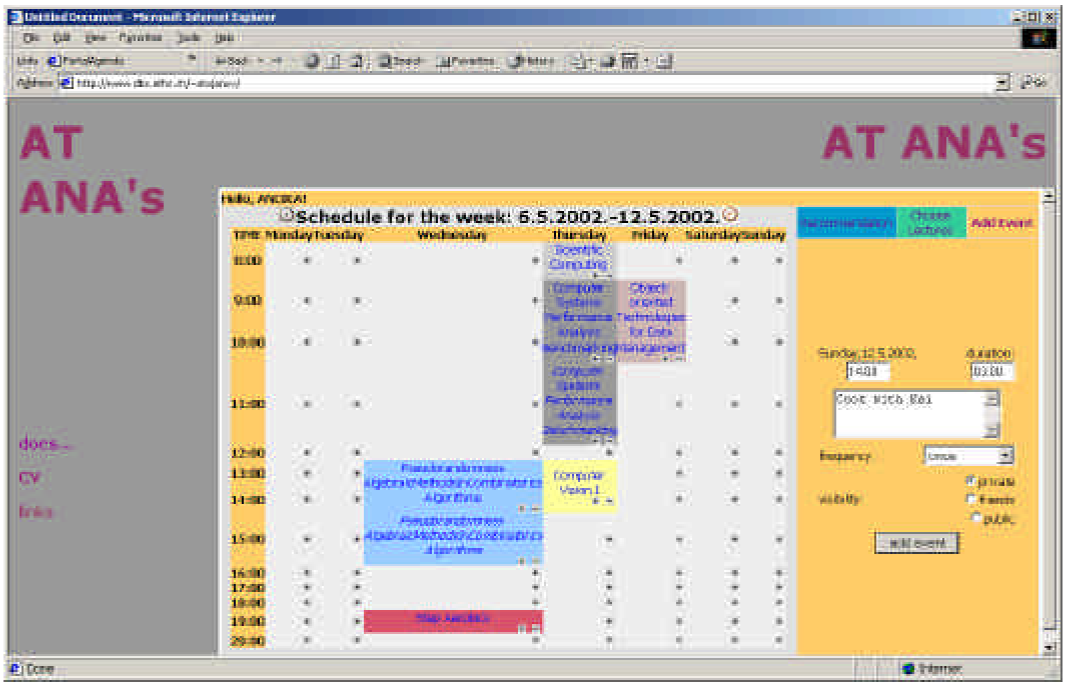

AT ANA's
AT ANA's
Schedule for the week: 6.5.2002.-12.5.2002.
TIME Monday Tuesday Wednesday Thursday Friday Saturday Sunday
Scientific Computing
Object oriented Techniques for Data management
Computer Vision I
Choose Lectures
Add Event
does...
CV
links
add event

Figure 5.14 A student portal

Figure 5.14 gives an impression of what that portal, in fact just one its part, should look like. In the middle of the page the week view on my agenda is shown. In the left, larger part, the week view to all the user duties is shown. Using the buttons, previous and next, it is possible to scroll through the weeks; Pluses and minuses allow customisation of the existing view i.e. adding or removing events. These events can be connected to some course and therefore be public and already scheduled (lectures, exercises, colloquium, exams) or student's own (whereas they can again be defined to be visible to the others). At the moment two more applications are developed. One allows course selection whereas the other recommends which courses to select (refer to sections 5.5.3, 5.5.4).

Let us take, for example, an ordinary student, graduate, a researcher (PhD or PhD student), an assistant or a professor at ETH; the things that all of them share

are the courses and timetables for those courses. These timetables, though in a part completely the same for large groups of people, are still very much different for each member of every group. All the members of the same group should have the matching parts in their timetables. Usually, that matching parts can be automatically presented to each of the members of that group. Initially, for students these groups are made compliant to the course of studies and the semester the student is in. But then, they change in conformance to the courses that specific students choose or visit. The selections, made by the student, can then be used for recommendations during the course selection process. There are various reasons why people visit some courses: some are motivated by the topic itself, others are motivated by the people that have chosen the same courses, yet others by the people who give those courses, or people simply choose courses regarding the way they fit in their existing schedule. For each student alone, it is always hard work to collect all this information, although it is publicly available. This was the motivation for developing a student portal. When a student enters the ***ETH World*** all this information should some way become available to him (her).

5.5.1 The CourseScheduler Database

The database *CourseScheduler* was developed for prototyping purposes. In the real environment this would probably be a federation of databases, each of which would contain just one part of the data (or metadata). Most of the data, indeed, already exists in various university databases and is used for completely different purposes. It is this combination of the existing data that can produce new knowledge and therefore improve interaction between the ETH and its individual members.

Figure 5.15 shows the schema of the underlying database. The database is modelled by having the following considerations in mind:

- *Users* can be either *Students* or *Lecturers* (professors, assistants).
- During different semesters, each *Student* can attend (*isAssignedTo*) different *LectureEvents* for various *Courses* that are available (*CoursesAvailable*) on the student's course of studies. If some student is not assigned to any of the events for a particular course, then he or she is, by definition, not assigned to the course itself.
- In various archives exists information about the courses the students have already listened and passed (*ExamsPassed*).
- *Courses* differ from semester to semester by the people who teach them (*Teaches*); by the topics presented (*Document*), and the type of the events that are connected to these courses (*Lecture*).
- Along with these university related events, each user, be it a *Student* or a *Lecturer*, can visit events that are not falling into this category i.e. sport, movie, … (*UserEvent*) and thus have as an entry in his or her timetable.

These events can be: private, visible only to student's friends, or public (refer to section 5.5.3).

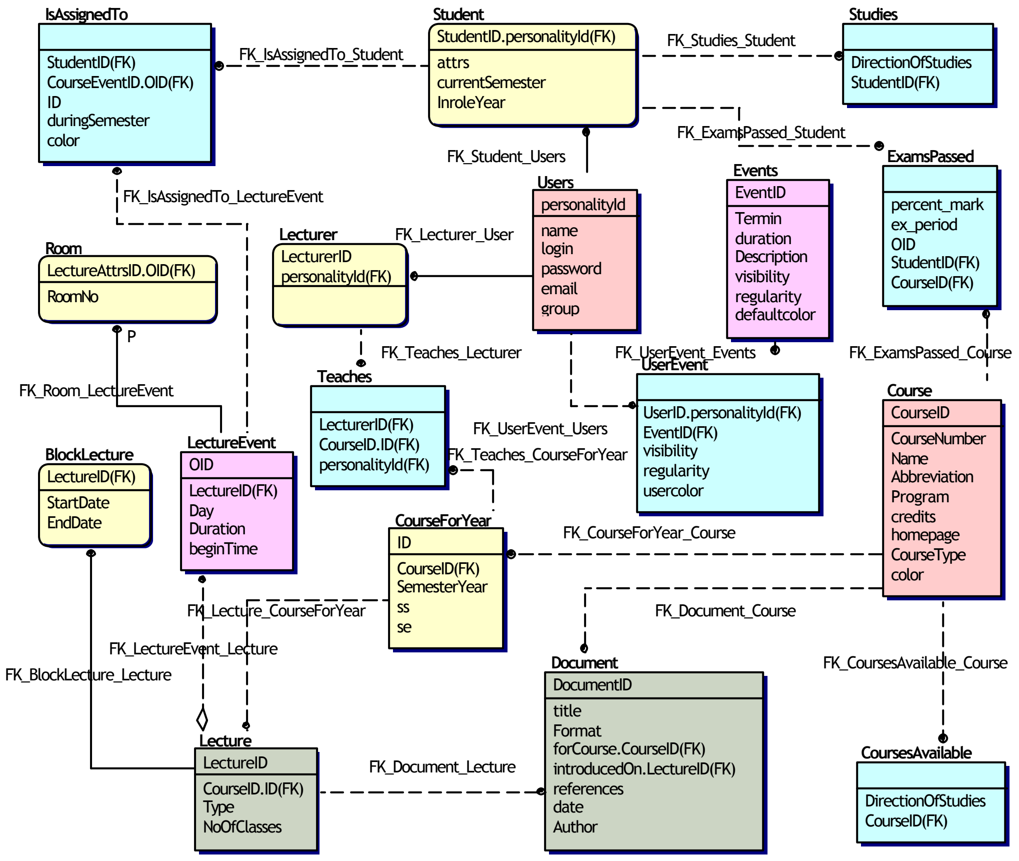
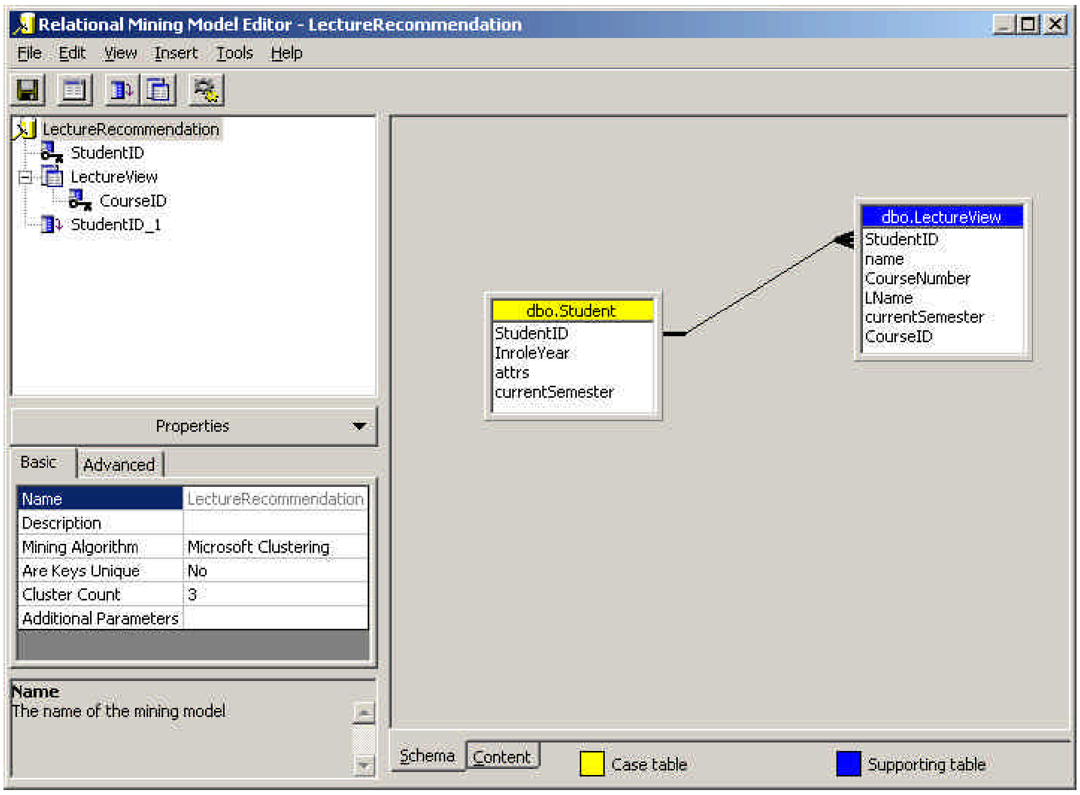


Figure 5.15 The scheme for the *Course Sheduler* relational database

5.5.2 Data Mining Model

Since most of the features of a data mining model are described in the previous chapters (refer to chapter 5.2.2), here just a short look will be taken at the way the model, used for this application, is presented and processed using ***Microsoft Analysis Manager***. Figure 5.16 shows the schema for the ***LectureRecommendation*** mining model and the way it could be created using `<advisor:create/>` tag.

When creating the mining model from the ***Microsoft Analysis Manager***, majority of the parameters from the `<connect/>` and `<rowsetData/>` tags are omitted, since they are already defined somewhere else in the data mining tool. The ***LectureView*** that is used, as a supporting (input) table, is created over tables *Users, Student, isAssignedTo, LectureEvent, Lecture, CourseForYear* and *Course* from the *CourseScheduler* database. It connects directly students and courses these students are assigned to. As mentioned earlier, the basic assumption in the *CourseScheduler* database is that a student cannot be assigned to a course if he or she is not assigned to any of the events for that course (e.g. lectures, exercises, tutorials ...).



Name	LectureRecommendation
Description	
Mining Algorithm	Microsoft Clustering
Are Keys Unique	No
Cluster Count	3
Additional Parameters	

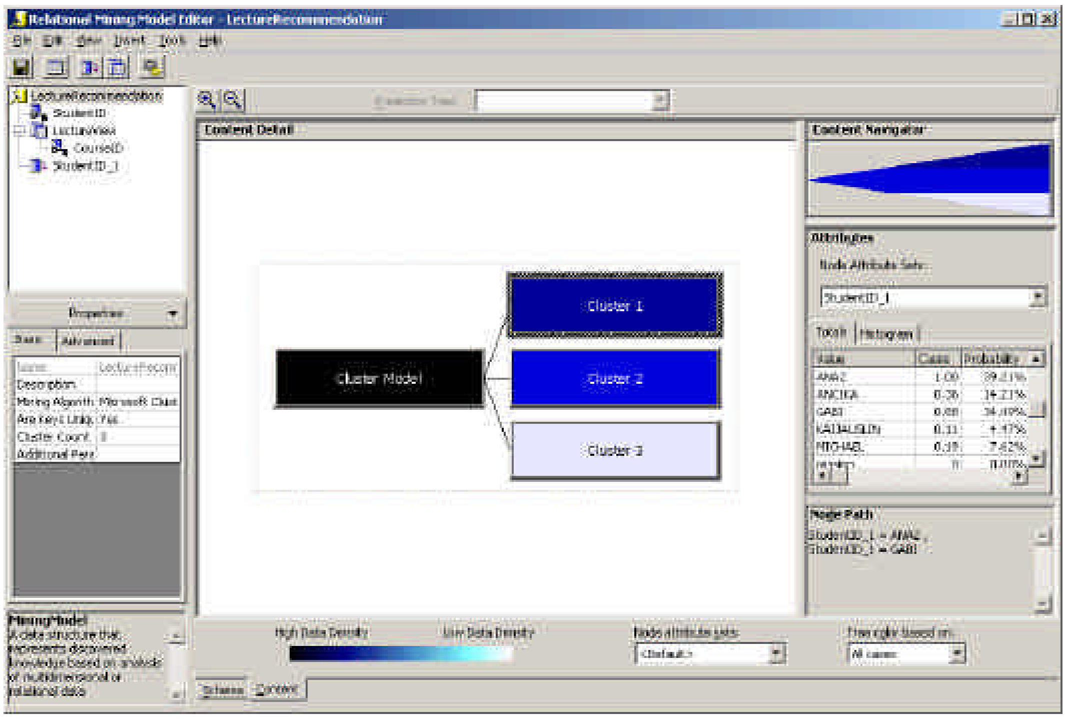
Name
The name of the mining model

Figure 5.16 The scheme for the *LectureRecommendation* DMM

Example 5.14:Create Mining Model

```
<harry:create model="LectureRecommendation" clusterCount="3">
   <harry:caseTable>Student</harry:caseTable>
   <harry:inputTable>StudentID</harry:inputTable>
   <harry:caseColumn>LectureView</harry:caseColumn>
   <harry:inputColumn>CourseeID</harry:inputColumn>
   <harry:connect>Location=GORILLAZ;Provider=msolap;
     MiningLocation=c:/CourseScheduler;</harry:connect>
   <harry:rowsetData>'SQLOLEDB.1','Persist Security Info = True;
   User ID= sunpress; Password=sunpress; Initial Catalog =
   CourseSchedule; Data Source=GORILLAZ</harry:rowsetData>
   <harry:sourceTable>'SQLOLEDB.1','Provider=SQLOLEDB.1;Integrated
   Security=SSPI; Persist Security Info=False;InitialCatalog =
   CourseSchedule; Data Source=GORILLAZ</harry:sourceTable>
</harry:create>
```

Figure 5.17 shows the content of the ***LectureRecommendation*** mining model. It can be perceived as a one level tree structure, where each cluster is one node in that tree coloured according to the data distribution. The ones that contain more cases are darker. As indicated in the sections 3.3.1.5 and 5.2.2, there are no clear boundaries between different clusters. So, each of the cases belongs partly to each of the clusters. The cluster, where the probability of belonging for the case is the greatest, is the one to be returned when the model is queried through ***OLE DB for Data Mining***.



Name	LectureRecom
Description	
Mining Algorith	Microsoft Clust
Are Keys Uniq	Yes
Cluster Count	3
Additional Para	

MiningModel
A data structure that represents discovered knowledge based on analysis of multidimensional or relational data

Value	Cases	Probability
ANA2	1.00	39.21%
ANCICA	0.36	14.21%
GABI	0.88	34.40%
KATALSLIN	0.11	4.17%
MICHAEL	0.19	7.62%
missing	0	0.00%

StudentID_1 = ANA2,
StudentID_1 = GABI

Figure 5.17 The content of ***LectureRecomendation*** DMM

5.5.3 *The InteractiveTimetable Application*

It is the main idea of the portal to gather all the information, in any format, that is valuable for one person, organization or any other kind of community arrange them in a personalized way for each of its users and allow every kind of that information intermixing. It would be more than optimistic to expect from one application to do that. Therefore, the portal is not just one application, it is an agglomeration of very different applications that by normally executing, and at the same time interacting among them, and with the rest of the world collect, categorize, process, organize and personalize the data. In order to make all these applications work together and show their heterogeneous outputs on one display, be it a desktop, laptop, PDA or a mobile phone. Hence, there has to exist one application that will organize just output, display and arrangement of all the other applications. In a word: "a manager-application". Here, it is the ***InteractiveTimetable*** application. Since this is just a prototype ***InteractiveTimetable*** is not really a very sophisticated application. For the moment it includes all the channels (applications) available and displays their output. Example 5.8 shows a part of this xJSP application.

As a result of the ***InteractiveTimetable*** application execution XHTML file that contains already 'styled' included applications is received. Applications are integrated into the ***InteractiveTimetable*** using a `<portal:include/>` tag. They are arranged in the way it is indicated by the XSL stylesheet assigned to the ***InteractiveTimetable*** application. For example, a stylesheet that produces an output as shown in the figure 5.14 is indicated in the example 5.16. Template part

`<xsl:copy-of select="./*" />` copies everything enclosed within the tags indicated by match clause of the template (e.g. `<xsl:template match="ChooseLectures">`). Since in this case it is a call to another application that application has to produce a correct XHTML output or the application ***InteractiveTimetable*** will return an error.

Example 5.15 extract from InteractiveTimetable.xjsp

```
<IntAge xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance">
<%@ taglib uri=http://www.vis.ethz.ch/~kai/da/taglib/portal prefix="portal" %>
<%@ taglib uri=http://jakarta.apache.org/taglibs/utility
 prefix="util"%>
...
<PortalAgenda>
    <portal:include uri="/apps/PortalAgenda.xjsp" />
</PortalAgenda>
<jsp:useBean id="appName" scope="page" class="portal.organizer.Name"/>
<jsp:useBean id="values" scope="page" class="java.lang.String[3]"/>
<util:If predicate="<%= appName.equals(values[0]) %>">
   <Recommend>
    <portal:include uri="/apps/Recommend.xjsp" />
   </Recommend>
</util:If>
<util:If predicate="<%= appName.equals(values[1]) %>">
   <ChooseLectures>
    <portal:include uri="/apps/ChooseLectures.xjsp" />
   </ChooseLectures>
</util:If>
<util:If predicate="<%= appName.equals(values[2]) %>">
   <AddEvent>
    <portal:include uri="/apps/addEvent.xjsp" />
   </AddEvent>
</util:If>
...
</IntAge>
```

For any other arrangement of the included applications, or omitting some of them or even including new ones it is necessary to make new XSL stylesheet.

Example 5.16 Stylesheet for InteractiveTimetable.xjsp

```
...
<xsl:template match="IntAge">
<HTML>
   <HEAD>
    <xsl:call-template name="css" />
    <link rel="stylesheet" href="appstyle.css" type="text/css"/>
   </HEAD>
   <BODY>
    <TABLE bgcolor="#e0e0e0" width="100%" height="80%">
    <TR>
    <TD width="80%"><xsl:apply-templates select="PortalAgenda" /></TD>
    <TD width="100" ALIGN="right" style="font-size:8pt;">
    <xsl:apply-templates select="ChooseLectures" />
    <xsl:apply-templates select="Recommend" />
    <xsl:apply-templates select="AddEvent" />
    </TD></TR>
    </TABLE>
   </BODY>
</HTML>
</xsl:template>
```

Example 5.16 Stylesheet for InteractiveTimetable.xjsp (continued)

```
<xsl:template match="PortalAgenda">
    <xsl:copy-of select="./*" />
</xsl:template>
<xsl:template match="ChooseLectures">
    <xsl:copy-of select="./*" />
</xsl:template>
<xsl:template match="Recommend">
    <xsl:copy-of select="./*" />
</xsl:template>
<xsl:template match="AddEvent">
    <xsl:copy-of select="./*" />
</xsl:template>
```

5.5.4 The Portal Agenda Application

The ***PortalAgenda*** is standalone web application that automatically generates a personal week view of all the important events for each portal user. Default view shows to a user all the events (typically university obligations) that can be derived from the available information on that particular user. User can, later on, customize the default view to his or her own. This, actually, means that he or she can add new or delete existing events and choose to share some of newly added events with friends or with everyone else (refer to section 5.5.6).

What makes this agenda different from most of the available ones is possibility to visualize parallel events. It is obvious that one person cannot be on more than one place at the same time, so this feature plays important role only in initial cases, when more people are in play and before the course list is customized. One could imagine the following scenario:

- Fifth semester Computer Science student enters the ***ETH World*** through his or her portal the first time in the current semester.
- Since the student has not decided yet which lectures to visit in the current semester, he or she chooses *show all possibilities* option. This agenda would then, have a look similar to the one shown by figure 5.18. Its basic disadvantage is the information overload, while an advantage it provides to user is a high degree of freedom.
- Student could then customize personal view by taking out the courses that are not of her or his interest.

The other possible situation is the one where most of the courses are fixed i.e. students are not allowed to choose them (for example, on first two years of Computer Science study here on ETH or even later the departments where credit system is not applied). Then majority of students have the same schedule and there exist just slight variations, between the exercise groups, for example.

The ***PortalAgenda*** application fetches the data from the database using the `<eventlist/>` tag and then formats it into XML using the `<weekview/>` tag.

Tags `<eventlist/>` and `<weekview/>` are implemented for this application needs, therefore they will be described in the following sections.

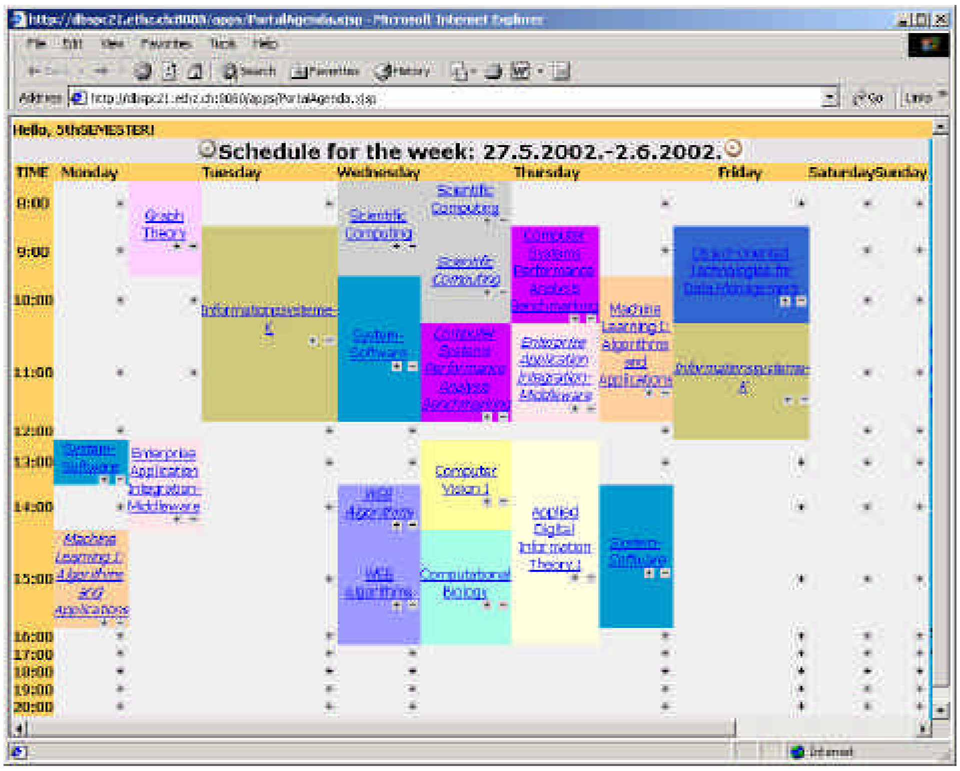

Hello, 5thSEMESTER!
Schedule for the week: 27.5.2002.-2.6.2002.
TIME
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Graph Theory
Scientific Computing
Computer Systems Performance Analysis Benchmarking
Object-Oriented Technologies for Data Management
Informationssysteme
System-Software
Machine Learning I: Algorithms and Applications
Enterprise Application Integration Middleware
Computer Vision I
Web Algorithms
Applied Digital Information Theory I
Computational Biology

Figure 5.18 Overview of available lectures in the 5th semester

5.5.4.1 EventListTag

Unlike the `<customEventList/>` tag, which generates a vector of objects of the `Event` class based on the `ResultSet` of the specified database query, the `<eventList/>` tag generates output vector, with already defined structure. The generated vector, as a result of the call to the `Events.getEvents()` method, contains events (instances of the `Event` class) that wrap records of the `ResultSets` of the queries, specified in the `FreeEvents`, `LectureEvents`, `SomeEvents` classes and so on. These classes inherit the `Events` class and override the `getEvents()` method, so that it returns events of the type specified by that query; in fact, the call to `LectureEvents.getEvents()` returns the vector of `Event` objects of the `Lecture` type… The `<eventList/>` tag allows, however, some adjusting. Having in mind that the list of events is different for each separate user, by indicating the `user` parameter it is defined what user is in question. Since the overview of events is limited onto one week, parameter `toDay` indicates the date in the week that should be rendered. It is assumed that the first day in the week is Monday and the last is Sunday. The third parameter is `scope`, which indicates visibility and the persistence of the resulting vector referenced by the `listOfEvents` attribute (refer to sections 4.4 and 5.4.1). Example 5.17 shows one utilization of the `<eventList/>` tag:

Example 5.17: one utilization of the <eventList/> tag

```
...
<util:If predicate="<%= evList==null %>" >
   <organizer:EventList userID="<%= userID %>" listOfEvents="evList"
   toDay="<%= TodayBean.getDate()%>" />
</util:If>
<organizer:outputXML src="evList"/>
```

As a consequence of generating the output with `<outputXML/>` tag, XML tree has very simple structure. The nodes in the tree can be of the different type, as shown by the result 5.18.

Result 5.18 a part of the application 5.17 output

```
<list>
   ...
   <Lecture>
        <usercolor>no</usercolor>
        <ltype>U</ltype>
        <dayNo>4</dayNo>
        <duration>2:00</duration>
        <defaultcolor>#0099cc</defaultcolor>
        <Homepage>cs.inf.ethz.ch/edu/37-201/</Homepage>
        <CourseID>7</CourseID>
        <Termin>2002-05-30 14:00:00.0</Termin>
        <type>Lecture</type>
        <day>Thursday</day>
        <OID>16</OID>
        <description>System-Software</description>
   </Lecture>
   <Free>
        <visibility>public</visibility>
        <usercolor>no</usercolor>
        <dayNo>5</dayNo>
        <duration>2:00</duration>
        <defaultcolor>#8080C0</defaultcolor>
        <regularity>weekly</regularity>
        <Termin>2002-05-31 19:15:00.0</Termin>
        <type>Free</type>
        <day>Friday</day>
        <OID>2</OID>
        <description>Volleyball</description>
   </Free>
   ...
</list>
```

It is obvious that the nodes `Lecture` and `Free` can contain different number of elements, where some of them exist only as a part of the `Lecture` node (for example, `Homepage`, `ltype`), while others exist only in the `Free` node (for example, `visibility`).

Transformation of this tree using the XSLT becomes complicated due to the following reasons:

- The XSLT template will contain a large number of `<xsl:if/>` (i.e. `<xsl:when/>`) elements, since for each node type appropriate stylesheet should be indicated.

- In order to make some other structure out of this flat one, for example table, the advanced knowledge in XSLT programming is required, since standard editors do not support complex operations.

5.5.4.2 *WeekViewTag*

Unlike the `<outputXML/>` tag, the `<weekview/>` tag transforms input vector into an XML tree, holding the structure defined by the *XSL-Scheme,* rendered on the figure 5.19 (xsd file is in the appendix B). This structure is then easily transformed into output format applying a XSL stylesheet (one stylesheet is shown by example 5.19). This tree, along with the information, contained in the source vector of events, contains some additional information about the layout. Similar to the `<eventlist/>` tag, `<weekview/>` tag takes, as parameters, the first and the last of the week to be displayed; hence, in the xJSP application it is called with:

```
<organizer:weekview src="eventList"
             firstDay="2002-05-27"
             lastDay="2002-05-31"/>
```

where it is assumed that the source vector is stored in the `pageContext` and the difference between dates indicated by the `firstDay` and `lastDay` parameters is not greater than 6 days. If these parameters are not indicated, it is assumed that all the members of the `eventList` vector should be displayed.

Root element `<Agenda/>` contains one `<Workdays/>` subtree and one or more `<TimeAgenda/>` elements. Each of the `<Day/>` elements in the `<Workdays/>` subtree along with the `CDATA` value, which stands for the name of the working day on the selected language, also contains the `date` and `maxevents` attributes that represent the date and the maximum number of overlapping events for that day. Due to HTML characteristics, an XML tree has to hold specific structure. Since the table is drawn raw by raw, events (*Time Based* – `TBEvent`) are grouped, first by days (`DayEvents`), and then by time they begin at i.e. each `<TimeAgenda/>` looks like (`TimeAgenda{begintime,DayEvents}`).

For the information visualization, consistent with needs and priorities of each user different XSL stylesheets can be used or the existing one can be adjusted. One possible stylesheet example is used in the example 5.19.

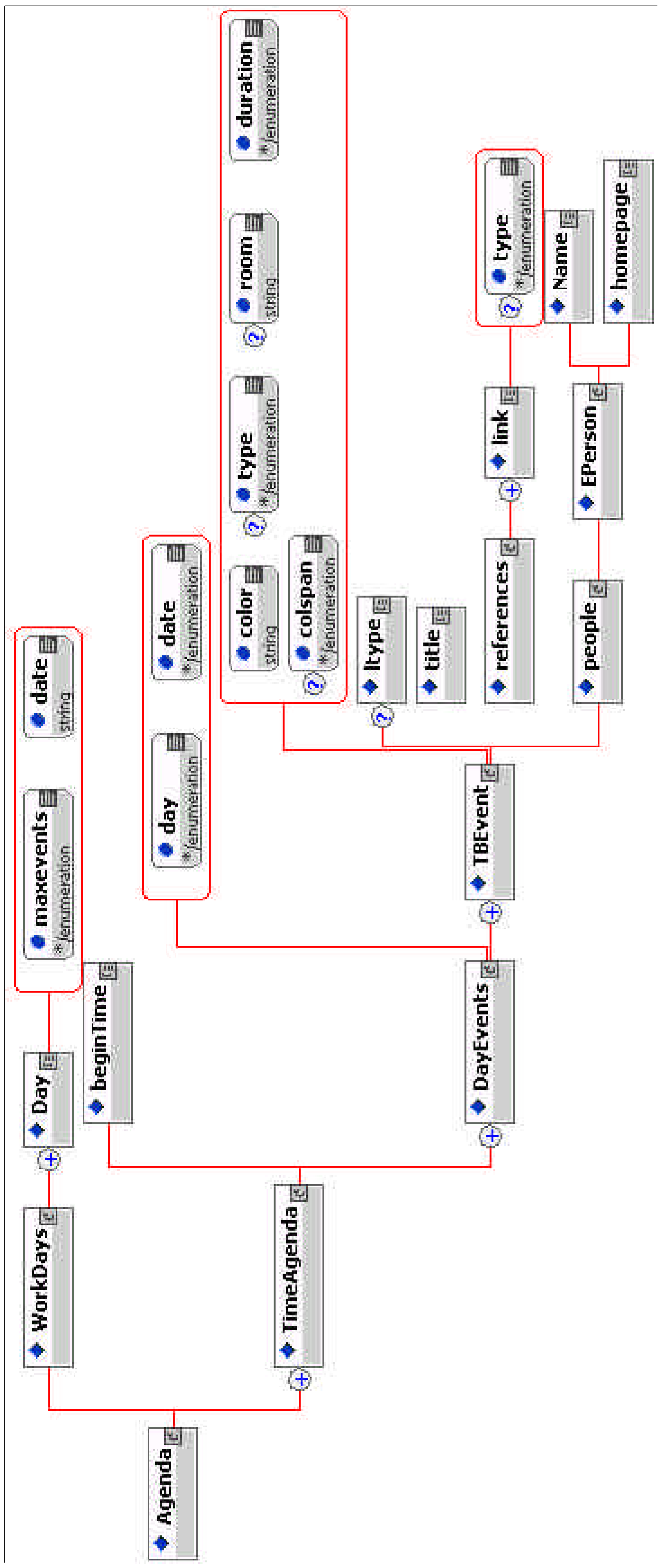


Figure 5.19 1 XML scheme for the XML tree generated using the <weekview/> tag

Example 5.19 Stylesheet for InteractiveTimetable.xjsp

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform ver-
sion="1.0">
<!—ukljucuje CSS file sa stilovima-->
<xsl:include href="css.xsl" />
<!—defines tip izlaznog file-a -->
<xsl:output method="html"/>
<!—defines promenljivu wdays -->
<xsl:variable name="wdays" select="Agenda/WorkDays"/>
<xsl:template match="/Agenda">
<atitle>
   <!—adds dynamic link to timetable for the previous week-->
   <a><xsl:attribute name="href">PortalAgenda.xjsp?previous=
   <xsl:value-of select="$wdays/Day[1]/@date"/></xsl:attribute>
   <img src="prev.gif" width="19" height="19" border="0" /></a>
   <!—adds title-->
        Schedule for the week:
        <xsl:value-of select="$wdays/Day[1]/@date"/> -
        <xsl:value-of select="$wdays/Day[last()]/@date"/>
   <!—adds dynamic link to timetable for the next week -->
   <a><xsl:attribute name="href">PortalAgenda.xjsp?next=
   <xsl:value-of select="$wdays/Day[last()]/@date"/></xsl:attribute>
   <img src="next.gif" width="19" height="19" border="0" /></a>
   <br/>
</atitle>
<table width="600" cellspacing="0" cellpadding="0" height="80%">
   <!—sets the first row of timetable (i.e. names of the days) -->
   <tr>
     <th>TIME</th>
     <xsl:for-each select="$wdays/Day">
        <th width="16%">
        <xsl:attribute name="colspan">
            <xsl:value-of select="number(@maxevents)"/>
        </xsl:attribute>
        <xsl:value-of select="."/>
        </th>
     </xsl:for-each>
   </tr>
   <!—sequential pass through, by beginTime, ordered file -->
   <xsl:for-each select="TimeAgenda">
    <tr>

      <!—in each pass writes start time-->
      <th><xsl:value-of select="beginTime"/>:00</th>

      <!—and calls named template columns -->
      <xsl:call-template name="columns">
         <xsl:with-param name="TimeAgenda" select="."/>
         <xsl:with-param name="fDay" select="1"/>
         <xsl:with-param name="event" select="1"/>
      </xsl:call-template>
      </tr>
   </xsl:for-each>
</table>
</xsl:template>
```

Izvod 5.19 Stylesheet za InteractiveTimetable.xjsp (nastavak)

```
<xsl:template name="columns">
   <xsl:param name="TimeAgenda"/>
   <xsl:param name="fDay"/>
   <xsl:param name="event"/>
   <xsl:if test="$fDay &lt;= 7">
      <xsl:variable name= "day" select="$ways/Day[$fDay]"/>
    <xsl:variable name="cs" select="$wdays/Day[$fDay]/@maxevents"/>
    <!—passes through all events,startin on day at beginTime-->
    <xsl:if test="$TimeAgenda/DayEvents[$event]/@day = $day">
        <xsl:for-each select="$TimeAgenda/DayEvents[$event]/TBEvent">
        <td>
    <!—and for each of them sets the background color, title,...-->
         <xsl:attribute name="bgcolor">
         <xsl:value-of select="@color"/>
         </xsl:attribute>
         <xsl:attribute name="rowspan">
         <xsl:value-of select="number(@duration)"/>
         </xsl:attribute>
         <xsl:attribute name="colspan">
         <xsl:value-of select="number(@cs)"/>
         </xsl:attribute>
         <a>
         <xsl:if test="ltype='U'">
            <xsl:attribute name="class">abc</xsl:attribute>
         </xsl:if>
         <xsl:if test="count(references/link)&gt;1">
            <xsl:attribute name="href">
            http://www.<xsl:value-of select="references/link"/>
            </xsl:attribute>
            <xsl:value-of select="title"/>
         </xsl:if>
         </a>
         <a>
         <xsl:if test="count(references/link)&gt;1">
            <xsl:attribute name="href">
            <xsl:value-of select="references/link[@type='r']"/>
            </xsl:attribute>
            <img src="minus.png" width="9" height="9" align="right"/>
         </xsl:if>
         </a>
         <a>
         <xsl:attribute name="href">
         <xsl:value-of select="references/link[@type='a']"/>
         </xsl:attribute>
         <img src="plus.png" width="9" height="9" align="right"/>
         </a>
        </td>
        </xsl:for-each>
      </xsl:if>
   <!—and then passes onto the next event -->
   <xsl:call-template name="columns">
      <xsl:with-param name="TimeAgenda" select="."/>
      <xsl:with-param name="fDay" select="$fDay+1"/>
      <xsl:with-param name="event" select="$event+1"/>
   </xsl:call-template>
   </xsl:if>
</xsl:template>
```

It is illustrated in the example 5.19 that there exist two types of the XSL templates, unnamed and named. Unnamed templates are requested at the moment when the processor, while dealing with XML file comes across an element with name specified in the `match` clause (`<xsl:template match="Agenda">` or `<xsl:apply-templates match="$wdays/Day">`). The named templates can also contain variables, which are passed on to template value (`<xsl:call-template name="columns">`). XSLT enables recursive template calls [XSLT].

5.5.4.3 ADDEVENT, REMOVEEVENT

"Portlet" ***AddEvent*** is a part of the ***PortalAgenda*** application, which provides the possibility to insert new events into the timetable. It is shown by the figure 5.14, that this xjsp application is called by clicking the "plus" link, anywhere in the table. The day, date and time are passed as arguments (for that the `DateBean` is used). Communication to the database is performed using tags from the `dbtags` tag library (refer to section 4.4.2.4). By utilizing `get` and `set` methods of the `DateBean` class the indicated parameters of the `PortalRequest` are accessed (see [J2EE]).

Example 5.20 AddEvent.xjsp

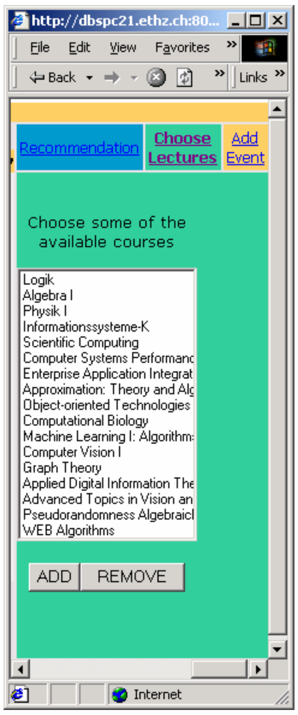
```
<?xml version="1.0" ?>
<AddEvent xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance">
   <%@ taglib uri="http://jakarta.apache.org/taglibs/dbtags" prefix="sql" %>
   <%@page extends="portal.framework.PortalJspPage" %>
   ...
   <jsp:useBean id="dBean" class="portal.taglib.organizer.DateBean"/>
   <jsp:setProperty name="dBean" property="day" param="day" />
   ...
   <newEvent>
   <day><%= dBean.getDay()%></day>
   <date><%= dBean.getDate()%></date>
   <time><%= dBean.getTime()%></time>
   </newEvent>
   <util:If predicate="<%=request.getParameter("submit")!=null%>" >
   <sql:connection id="conn">
        <sql:url>...</sql:url>
        <sql:driver>...</sql:driver>
   </sql:connection>
   <sql:statement id="stmt1" conn="conn">
        <sql:query>
          select max(EventID) from Events
        </sql:query>
        <sql:resultSet id="rset1" >
          <sql:getNumber position="1" to="evid" scope="page"/>
        </sql:resultSet>
   </sql:statement>
   ...
   <sql:preparedStatement id="stmt2" conn="conn">
        <sql:query><%@ include file="addEvent.sql" %></sql:query>
        <sql:execute>
        <sql:setColumn position="1"><%=dBean.getSQLT()%></sql:setColumn>
        ...
        </sql:execute>
   </sql: preparedStatement>
   <% response.sendRedirect("PortalAgenda.xjsp?appName=Recommend");%>
   <sql:closeConnection conn="conn"/>
   </util:If>
</AddEvent>
```

Example 5.20 shows that the ***AddEvent*** application is comprised of two parts. First part, before the condition (`<util:If>`), is executed every time the application is called, while the second part, which includes writing to the database (tags `<statement>` and `<preparedStatement/>`), is executed only if the "add event" button is pushed. After the execution of this, conditional part, the redirec tion to other application is performed; hence, this xJSP application has to extend the `portal.framework.PortalJspPage` class (refer to section 4.3.2).

Opposed to ***AddEvent***, the ***RemoveEvent*** application removes the selected event from the list of actual events for the given week. The selected event can be of any of the previously mentioned types (here, `LectureEvent` or `FreeEvent`).

5.5.5 The ChooseLectures Application

The ***ChooseLectures*** application presents to the identified user (in this case student) the list of courses available in the current semester, based on data available on that user (here: course of studies and current semester). The strudent is allowed to choose the courses from the presented list that he'd like to attend. From the list of all the courses available in the current semester, the ones, that the user had passed are removed (the filter tag is used, refer to appendix C), and the result is presented to the user. When the student selects the course and adds it to the list of his/her own courses, it is immediately shown in the timetable (application ***PortalAgenda***) that shows the week program of student's responsibilities. Similar to the ***AddEvent*** application, this application is comprised of two parts that are executed depending ob the fact whether the selection was or was not performed. Unlike the other application, here both parts communicate to the database, where in the first part (which is always executed) only reading from the database is performed, while in the other part database is also accessed for writing i.e. connecting all the events related to the particular course with the active user. Figure 5.20 shows one possible manifestation of this application.



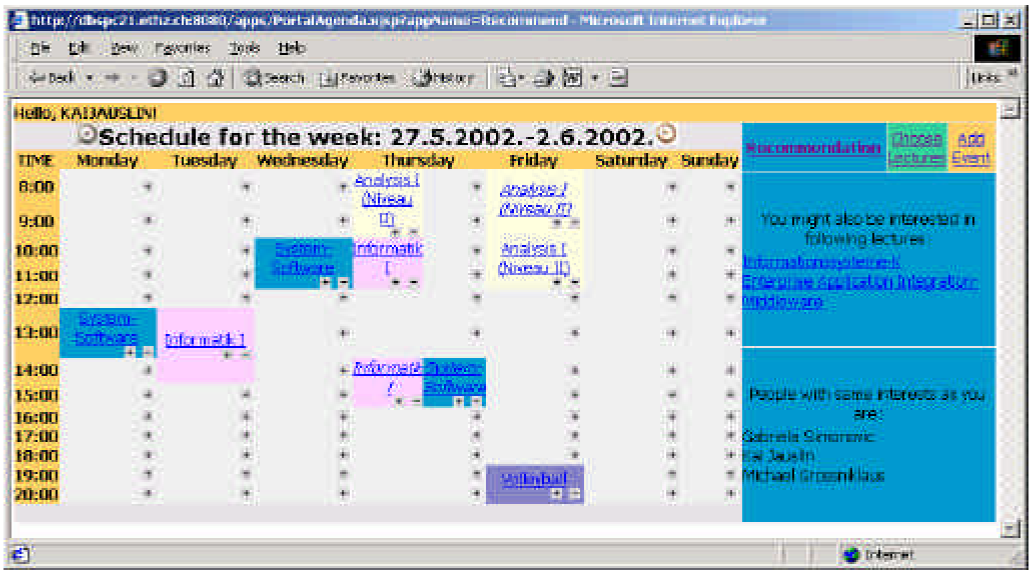
Choose some of the available courses

Logik
Algebra I
Physik I
Informationssysteme-K
Scientific Computing
Computer Systems Performanc
Enterprise Application Integrat
Approximation: Theory and Alç
Object-oriented Technologies
Computational Biology
Machine Learning I: Algorithm:
Computer Vision I
Graph Theory
Applied Digital Information The
Advanced Topics in Vision an
Pseudorandomness Algebraicl
WEB Algorithms

Figure 5.20 The ***ChooseLectures*** application

5.5.6 The Recommend Application

This application, as previously mentioned, recommends the courses. By using the tags from the ***advisor*** tag library the cluster, the identified user, in the indicated interest space belongs to. In this application user interests are represented by the courses that that user selected once. Then, that active user is presented all the features of the cluster, that user belongs to, and are not the features of his own. In fact, the student is presented the courses selected by the other student that belong to same cluster. As some kind of explanation, student is presented all other students, which belong to the same cluster. Clicking the link, the name of the user, it is possible to send him an e-mail. By clicking the *chat* button, beside student's name, the *real-time* communication, with another student from the same cluster, is started (using the ICQ client). The privacy violation that exists in the prototype can be solved by using aliases and groups. The number of clusters is automatically tuned to the number of users (for example, first integer value greater that the square root of the total number of students), but can also be manually set in the application using the `retrain` tag. However, this way the application performance is significantly reduced, since the time for retraining is required.


Hello, KAIJAUSLINI

Schedule for the week: 27.5.2002.-2.6.2002.

You might also be interested in following lectures:
Informationssysteme
Enterprise Application Integration
Middleware

People with same interests as you are:
Gabriela Simonovic
Kai Jauslin
Michael Groesnklaus

Figure 5.21 The *Recommend* application

Example 5.21 Recommend.xjsp

```
...
<recommendations>
   ...
   <harry:getModelInfo id="MI" forModel="LectureRecommendation"/>
   <harry:query model="LectureRecommendation"
   caseID="<%=userID%>" result="RS" />
   <hermione:filter type="SELECT" key="StudentID"
   value="<%= userID %>" result="selected">
    <hermione:source id="SameInterests" >
   <harry:getColumns model="LectureRecommendation" queryRS="RS" />
    </hermione:source>
   </hermione:filter>
   <hermione:filter type="DIFFERENCE" key="CourseID" result="filtered">
    <hermione:source id="SameInterests"/>
    <hermione:source id="selected"/>
   </hermione:filter>
   <hermione:outputXML src="filtered"/>
   <harry:getColumns model="LectureRecommendation" queryRS="RS"
   attribute="name"/>
</recommendations>
```

Figure 5.21 shows one possible interface for this application (blue part). While the upper part contains the list with recommended courses the lower contains other users "similar" to the identified one. When the link of some course from the list is clicked, it is attached to the user's list of selected courses, and to his timetable, which is immediately noticeable in the ***PortalAgenda*** application. In the example 5.21 the part of the xjsp code for this application is shown.

In this example, both ***advisor*** (namespace `harry`) and ***organizer*** (namespace `hermione`) tag libraries are used. In order to be able to generate recommendations application needs to get all the information (tag `<getModelInfo/>`) on the clustering model ***LectureRecommendation*** (refer to section 5.5.2), and then access the information on cluster that the active user (identified by `userID`) belongs to (tag `<getColumns/>`). After that, the courses selected by the active user are filtered out from all the courses that describe the identified cluster (`SELECT` and `DIFFERENCE` filters); and the active user is presented the others, as well as the other users belonging to the same cluster (`<getColumns/>` with the `name` attribute of the `inputTable`). As already mentioned in the section 5.3 (see also the appendix B), this tag provides access to all the table columns used for clustering (`inputTable`). Hence, application programmer has a higher degree of freedom when knowing all these attributes (belonging to `caseTable` and `inputTable`).

6 Conclusion

6.1 University Portals

The World-Wide-Web has grown to be all-purpose tool, used in everyday work, for information retrieval and business execution processes. Web is responsible for the revolutionary changes in both, the function and the availability of information. Almost any task, even the simplest web search, the basic work or academic task, requires coordination of myriad of information sources, processes and data – not to mention the integration of a multiple of desktop, enterprise, and web technologies. To date on the web there are over three billion of web pages, and search engines (horizontal portals) are still ineffective in a try to reduce list of hits onto acceptable size, and in the given context, rank results in a meaningful manner. They do not solve the problem that appears during navigation process and leads to user disorientation in the hyperspace, neither multimedia contents problem. Opposed to horizontal, vertical, in fact, corporate portals support basic transformation of the user's view to the information management process within the organization. Their role includes, not only helping particular users to find the sense in the piles of available data, but also help them keep that knowledge. Institutional information portal, as a subgroup of vertical portals, represents an "application" that provides its users with unique, intuitive and personalized access and integration of information specific to that institution, stored in the internal databases and systems, with the information from the outside world. The browser represents the basic and ubiquitous client for the portal, which is a universally available framework. As such, portal represents the shift in the philosophy of the institution, which utilizes it, when providing services, as well as a great shift to the user – oriented design. In the portal structure the user is a "star" and all the contents and all services are adjusted to that basic idea. In particular, old-fashioned time/place/content predetermined education moves toward just-in-time/at work-place/customized/on-demand process.

The ***ETH World*** portal framework is just one small step towards what real university portal should represent. Unlike ***uPortal*** (refer to section 4.6.3), which is predominantly information supply oriented, the focus of this project is a bit shift to both, education and socialization processes. The recommendation module holds in that the main part. As it is obvious from the ***Recommend*** application this module enables, not only recommendation generation, but also virtual community formation. The problem of detecting virtual communities and social networks has been examined in [KAU97]. Numerous studies have shown that one of the most effective channels for dissemination of information and expertise within an organization is its informal network of collaborators, colleagues and friends. Part of the success of these networks can be attributed to the "six degrees of separation" phenomena, which states that the distance between two individuals in terms of direct personal relationships is relatively small. An equally important factor is that there are limits as to the amounts and kind of information that a person is able or willing to make available to the public at large. Hence, the main objective of the ***Referral Web*** project, described in [KAU97] consists in detection of the existing social networks with the goal of recommendation generating and not initiation for forming new ones. In the computer era, however, this last aspect gains on its importance since the alienation has become very broad problem. This thesis was a try to reduce the alienation existing in the university environment.

6.2 Possible Improvements

In the fourth and fifth chapter it is explained that the goal of the developed module was implementing the additional functionality to the ***ETH World*** portal framework: a recommendation generation. Due to the experimental nature of the whole framework, this module was developed as a tag library, which for communication to database and the *data mining* provider uses already existing *3rd party* components. These components have brought some limitations into implementation: used ***dbtags*** tag library does not support transactions, and ***JACOB*** bounds the platform to Windows, which is the consequence of using ***COM***.

Employed *data mining* provider ***Microsoft Analysis Services*** limits the *data mining* algorithms on decision trees, and EM clustering. One of the biggest weaknesses of this type of clustering is that the number of clusters must be fixed initial to the training process. Implemented tag library performs clustering based only on presence or absence of particular dimensions. In fact all the dimensions have same weights. The extension on weighting scheme is straightforward; it requires adding new attribute to the `<create/>` tag. The use of other *data mining* algorithms is limited to those implemented by the providers, which support ***OLE DB for Data Mining***. Module can be extended in a way that it accepts queries formulated in ***PMML*** (*Predictive Model Markup Language*) [PMML]. These features would provide application programmer more freedom, which implies the need for better notion of *data mining* algorithms. Knowing that, in the meantime, a new standard tag library - ***jstl*** appeared what implements all ***dbtags*** functionalities, and transactions, one of the next goals would be to adjust the *handler* class for `<customEventList/>` tag, in such manner that it utilizes the classes from this tag library. Due to rise in popularity of ***uPortal*** and ***Cocoon*** frameworks, an interface to them would be useful. The latter would make this tag library available to a number of portal frameworks (since the ***sunSpot*** portal engine for ***Cocoon*** appeared in the second quarter of 2002).

6.3 Utilization

One of possible use of the developed libraries ***advisor*** and ***organizer*** is illustrated in the sections 5.5, 5.6 and 5.7. In the section 5.1 it is also conveyed that the ***advisor*** tag library can be used for clustering other kinds of contents, too.

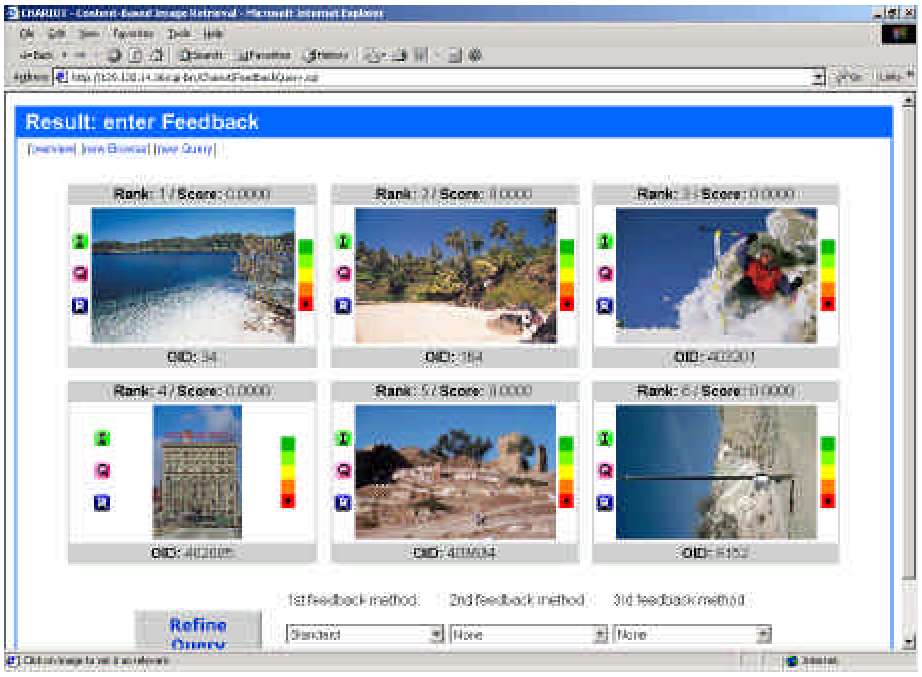


Figure 6.1 The *Chariot* system for image retrieval

In this section a short explanation on how it is possible to extend existing *Chariot* Image Retrieval system [CH00], implemented on Tk/tcl platform (figure 6.1) is given. Since this system does not have the user, only session identification, it is possible to select anonymous profiles. The profiles comprise a session identifier, rated image identifiers and the ratings, i.e. a `log` relation should have the following format `(session_id, image_id, rating)`. The process of clustering is, then performed based on `session_id`. The active session can be represented in the following profile format `(`$\texttt{log}_1$`,` $\texttt{log}_2$… $\texttt{log}_n$`)`, where n is a number of rated images. At the moment, in order to be able to pose a query data has to be stored in the database, with `session_id` as key attribute. The key is, as an active session identifier, sent to some kind of ***Recommend*** application (e.g. parameter `case_id` in the `<query/>` tag). The application, as a result, gives identifiers of the first k highest rated images from the cluster that the active session belongs to. That way a CF recommendation is formed. These results can be later combined with those got using an IR method, for example, by showing only those images that are elements of both lists.

6.4 Usability of the Developed Platform

As an outcome of information technology boom in the last twenty years of the twentieth centaury motto: "if you are not on web then you don't exist". Everybody wanted to "get out" on the web and each sphere of contemporary life received its *e* variant (*e-mail, e-business, e-banking, e-learning…*). Off course, due to using various tools and diverse amounts of time, many technologically different solutions, for a web "break through" appeared. The results are: enormous amount of knowledge on the Internet and everywhere else, but shortage of efficient tools to access that data. Portals help its users to find and use this information, which they need in the given context. Since the circumstances change, while the user moves in space, prefix *e* changes to *m* (*mobile*: *m-banking, m-business*...), and then to *p* (*pervasive*). However, the basic feature of a number of portals stays that they show user just what he/she has explicitly requested. Still, a user very often during the task execution needs much more than he is actually able to formulate. Thus, the recommendation facility in the portal gains its significance. It provides the user with the chance to see the world from somebody else's point of view.

When the first prototype of the ***ETH World Portal*** framework was implemented, there was no other portal solution, which could support the basic personalization condition, complete separation of the programming logic and the application layout. The trends in the development of web systems, owing to acknowledgement of XML for standard data transfer format and the fact that Java is secure programming platform, as well to W3C activities, lead to Java/XML frameworks (stable ***.NET*** appeared a bit later). Current state on the market shows that the choice was justified. At the moment there are lots of commercial deployment frameworks, which help authors in creating web applications (***iViewStudio*** …). Some of them appeared even before the work on this thesis started. Having in mind their features, for example price, the decision when starting this thesis was to work with standards. Hence, the ***JSP Taglib*** technology is used. Having in mind that, at that specific point in time, still no complete portal solution existed (***SAPPortal*** appeared in the fourth quarter of 2001); the easiest thing to do was extending the ***ETHWorld Portal*** prototype. Since the development framework does not exist, application development is done manually, which can be a bit tricky. However, each of the developed tags implements one functionality; hence their utilization is completely intuitive. As a whole, the system is pretty much unscalable due to DOM utilization for the base model. If this characteristic is considered, it is much more convenient to use ***Cocoon/sunSpot*** combination (even though XSP was not accepted by W3C as a standard). As far as I know, at the moment there is no other "free" recommendation module, so that ***advisor*** can finish a part of the job. In order to achieve the possibilities of commercial modules, like ***NetPerceptions*** (***Amazon.com***) is, a lot of work is required, including the implementation of other *data mining* techniques. But it goes out of the boundaries of this thesis, which only wanted to prove that all this was possible to do.

APPENDIX A

SAX Filter and SAX XML Generator

SimpleSAXFilter.java

```
import org.xml.sax.helpers.XMLFilterImpl;
import org.xml.sax.XMLReader;
import org.xml.sax.SAXException;
import org.xml.sax.Attributes;
import sax.helpers.AttributesImpl;
import org.xml.sax.helpers.DefaultHandler;
public class SimpleSAXFilter extends XMLFilterImpl{
public SimpleSAXFilter (){}
public SimpleSAXFilter (XMLReader parent){
   super(parent);
}
// Filters start-element events for dbconnection element
public void startElement (String uri, String localName,String qName, Attributes atts)
   throws SAXException {
   boolean filter=false;
   if (qName.equals("dbconnection")) {
     super.startElement(uri, localName, qName, null);
     int len = atts.getLength();
     for (int i = 0; i < len; i++) {
      super.startElement(atts.getURI(i), atts.getLocalName(i), atts.getQName(i),
      null);
      super.ignorableWhitespace(atts.getValue(i).toCharArray(),0,
      atts.getValue(i).length());
      super.endElement(atts.getURI(i), atts.getLocalName(i), atts.getQName(i));
     }
     filter=true;
   }
   if (!filter) super.startElement(uri, localName, qName, atts);
}
//Filters end-element events for dbconnection element
public void endElement (String uri, String localName, String qName)
   throws SAXException{
   if (qName.equals("dbconnection")) {}
   super.endElement(uri, localName, qName);
}
}
```

This filter is used by the class `SAX2Writer` to convert all the attributes of the `<dbconnection/>` element to its child elements.

SAX2Writer.java

```
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import sax.helpers.AttributesImpl;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
/**Example SAX2 writer. This example illustrates how to register
 * SAX2 ContentHandler that receives messages from SAXParser and
 * prints document which is parsed.*/
public class SAX2Writer extends DefaultHandler {
// constants
private static final String
DEFAULT_PARSER_NAME = "org.apache.xerces.parsers.SAXParser";
private static boolean setValidation= false; //defaults
private static boolean setNameSpaces= true;
private static boolean setSchemaSupport = true;
protected PrintWriter out;
protected boolean canonical;
// Default constructor
public SAX2Writer(boolean canonical)
   throws UnsupportedEncodingException {
     this(null, canonical);
   }
protected SAX2Writer(String encoding, boolean canonical)
   throws UnsupportedEncodingException {
   if (encoding == null) {
   encoding = "UTF8";
   }
   out = new PrintWriter(new OutputStreamWriter(System.out, encoding));
   this.canonical = canonical;
}
// prints the output of SAX messsages
public static void print(String parserName, String uri, boolean canonical)
{
   try {
     DefaultHandler handler = new SAX2Writer(canonical);
     XMLReader parser = new SimpleSAXFilter(
      (XMLReader)Class.forName(parserName).newInstance());
     parser.setContentHandler(handler);
     parser.setErrorHandler(handler);
     parser.parse(uri);
   }catch (Exception e) {
     e.printStackTrace(System.err);
   }
}
// Processing instruction.
public void processingInstruction(String target, String data) {
   out.print("<?");
   out.print(target);
   if (data != null && data.length() > 0) {
     out.print(' ');
     out.print(data);
   }
   out.print("?>");
   out.flush();
}
```

SAX2Writer.java (continued)

```
// Start document.
public void startDocument() {
   if (!canonical) {
     out.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
     out.flush();
   }
}
// Start element.
public void startElement(String uri, String local,
String raw,Attributes attrs) {
   out.print('<');
   out.print(raw);
   if (attrs != null) {
     attrs = sortAttributes(attrs);
     int len = attrs.getLength();
     for (int i = 0; i < len; i++) {
        out.print(' ');
        out.print(attrs.getQName(i));
        out.print("=\"");
        out.print(normalize(attrs.getValue(i)));
        out.print('"');
     }
   }
   out.print('>');
   out.flush();
} // startElement(String,String,String,Attributes)
// Characters.
public void characters(char ch[], int start, int length) {
   out.print(normalize(new String(ch, start, length)));
   out.flush();
}
// Ignorable whitespace.
public void ignorableWhitespace(char ch[], int start, int length) {
   characters(ch, start, length);
   out.flush();
}
// End element.
public void endElement(String uri, String local, String raw) {
   out.print("</");
   out.print(raw);
   out.print('>');
   out.flush();
}
// ErrorHandler metods
// Warning.
public void warning(SAXParseException ex) {
   System.err.println("[Warning] "+getLocationString(ex)+": "+
   ex.getMessage());
}
// Error.
public void error(SAXParseException ex) {
   System.err.println("[Error] "+getLocationString(ex)+": "+
   ex.getMessage());
}
// Fatal error.
public void fatalError(SAXParseException ex) throws SAXException {
   System.err.println(
   "[Fatal Error] "+getLocationString(ex)+": "+ex.getMessage());
   throw ex;
}
```

SAX2Writer.java (continued)

```
// Returns location of String.
private String getLocationString(SAXParseException ex) {
   StringBuffer str = new StringBuffer();
   String systemId = ex.getSystemId();
   if (systemId != null) {
     int index = systemId.lastIndexOf('/');
     if (index != -1) systemId = systemId.substring(index + 1);
     str.append(systemId);
   }
   str.append(':');str.append(ex.getLineNumber());
   str.append(':');str.append(ex.getColumnNumber());
   return str.toString();
}
// Normalizes string.
protected String normalize(String s) {
   StringBuffer str = new StringBuffer();
   int len = (s != null) ? s.length() : 0;
   for (int i = 0; i < len; i++) {
    char ch = s.charAt(i);
    switch (ch) {
     case '<': { str.append("&lt;"); break;}
     case '>': { str.append("&gt;"); break;}
     case '&': { str.append("&amp;"); break;}
     case '"': { str.append("&quot;"); break;}
     case '\r':
     case '\n': {
      if (canonical) {
        str.append("&#");
        str.append(Integer.toString(ch));
        str.append(';');
        break;
      }
     }
     default: {str.append(ch);}
    }
   }
   return str.toString();
}
// Returns sorted list of attributes.
protected Attributes sortAttributes(Attributes attrs) {
   AttributesImpl attributes = new AttributesImpl();
   int len = (attrs != null)?attrs.getLength():0;
   for (int i = 0; i < len; i++) {
     String name = attrs.getQName(i);
     int count = attributes.getLength();
     int j = 0;
     while (j < count) {
      if (name.compareTo(attributes.getQName(j)) < 0) {
        break;
      }
      j++;
     }
     attributes.insertAttributeAt(j, name,
                 attrs.getType(i), attrs.getValue(i));
   }
   return attributes;
}
/** Main program entry point. */
public static void main(String argv[]) {
   if (argv.length == 0) { System.exit(1);}
   boolean canonical  = false;
   String  parserName = DEFAULT_PARSER_NAME;
   print(parserName, argv[0], canonical);
}
```

APPENDIX B

XML Scheme for dynamically generated XML

WeekView.XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:db=" http://www.vis.ethz.ch/~kai/" elementFormDefault="qualified">
   <xsd:element name="Agenda">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="WorkDays"/>
        <xsd:element ref="TimeAgenda" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="WorkDays">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Day" type="DayType" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:complexType name="DayType">
    <xsd:complexContent>
     <xsd:extension base="DayName">
      <xsd:attribute name="maxevents">
         <xsd:simpleType>
            <xsd:restriction base="xsd:int">
               <xsd:minInclusive value="1"/>
               <xsd:maxInclusive value="9"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="date" type="xsd:string"/>
     </xsd:extension>
    </xsd:complexContent>
   </xsd:complexType>
   <xsd:simpleType name="DayName" type="db:dayname"/>
   <xsd:element name="TimeAgenda">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="beginTime" type="xsd:int"/>
        <xsd:element ref="DayEvents" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="DayEvents">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="TBEvent" maxOccurs="unbounded"/>
```

WeekView.XSD(continued)

```
      </xsd:sequence>
      <xsd:attribute name="day" type="DayName"/>
    <xsd:attribute name="date" type="xsd:string"/>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="TBEvent">
    <xsd:complexType>
      <xsd:attribute name="color" ref="xsd:color"/>
      <xsd:attribute name="type" type="db:type" use="optional"/>
      <xsd:attribute name="room" type="xsd:string"  use="optional"/>
      <xsd:attribute name="duration">
      <xsd:simpleType>
        <xsd:restriction base="xsd:int">
         <xsd:minInclusive value="1"/>
         <xsd:maxInclusive value="9"/>
        </xsd:restriction>
      </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="colspan" type="xsd:int" use="optional" value="1"/>
      <xsd:sequence>
        <xsd:element ref="ltype" use="optional"/>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element ref="references"/>
        <xsd:element ref="people"/>
      </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="references">
    <xsd:sequence>
      <xsd:element ref="link" maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:element>
   <xsd:element name="link">
    <xsd:simpleType>
      <xsd:extension base="xsd:anyURI" >
        <xsd:attribute name="type" type="linkType" use="optional"/>
      </xsd:extension>
    </xsd:simpleType>
   </xsd:element>
   <xsd:simpleType name="linkType">
     <xsd:enumeration value="a"/>
     <xsd:enumeration value="r"/>
   </xsd:simpleType>
   <xsd:element name="people">
    <xsd:sequence>
     <xsd:element ref="EPerson" maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:element>
   <xsd:element name="EPerson">
    <xsd:sequence>
     <xsd:element name="Name" type="xsd:string"/>
     <xsd:element name="homepage" type="xsd:anyURI"/>
    </xsd:sequence>
   </xsd:element>
</xsd:schema>
```

APPENDIX C

Code for xJSP application ChooseLectures

ChooseLectures.xjsp

```
<ChooseLectures xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance">
<%@ taglib uri="http://jakarta.apache.org/taglibs/dbtags" prefix="sql" %>
<%@ taglib uri="http://jakarta.apache.org/taglibs/utility" prefix="util"%>
<%@ taglib uri='agenda' prefix='hermione' %>
<%@page extends="portal.framework.PortalJspPage" %>
<%@page import="portal.personalization.*, portal.framework.*, java.util.*,
 java.sql.ResultSet"%>
<%@page import="portal.taglib.organizer.*" %>
   <hermione:EventList userID="<%= userID %>"  listOfEvents="evList" />

   <!--from list of all events shown extracts those of type Lecture-->
   <hermione:filter type="select" key="type" value="Lecture" result="Lattended">
     <hermione:source>evList</hermione:source>
   </hermione:filter>

   <!-- then finds out all distinct Courses these Lectures belong to-->
   <hermione:filter type="distinct" key="CourseID" result="chosen">
     <hermione:source>Lattended</hermione:source>
   </hermione:filter>

   <!-- for particualar student discovers all courses available to assign to-->
   <hermione:customEventList id="cel" userID="<%= userID %>"  type="Course"
    listOfEvents="available" >
     <sql:query>
        <%@ include file="available.sql" %>
     </sql:query>
     <sql:execute>
        <sql:setColumn position="1"><%= userID %></sql:setColumn>
        <sql:setColumn position="2"><%= userID %></sql:setColumn>
     </sql:execute>
   </hermione:customEventList>

   <!-- and then from all courses from which student could choose takes out ones
   (s)he's chosen already -->
   <hermione:filter type="difference" key="CourseID" result="choosable">
     <hermione:source>available</hermione:source>
     <hermione:source>chosen</hermione:source>
   </hermione:filter>

   <hermione:customEventList id="ud" userID="<%= userID %>" type="UserData"
    listOfEvents="uData" >
     <sql:query>
        SELECT * FROM Student WHERE StudentID = ?
     </sql:query>
     <sql:execute>
        <sql:setColumn position="1"><%= userID %></sql:setColumn>
     </sql:execute>
   </hermione:customEventList>
```

ChooseLectures.xjsp (continued)

```
   <chooseList>
   <hermione:outputXML src="choosable"/>
     <size><%= ((Vector)pageContext.findAttribute("choosable")).size() %></size>
   </chooseList>
   <%  String[] values = ((PortalRequest)request).getParameterValues("select");
   %><util:If predicate="<%= values != null %>">
   <% //if select parameter of the request is not null (which means that it is called
   from addLecture then
     //all chosen lectures are added to the database (stmt4)
     int currentSemester=0;
     values = ((PortalRequest)request).getParameterValues("select");
     StringBuffer set=new StringBuffer();
     set.append("('");set.append(values[0]); set.append("'");
     for(int i=1;i<values.length; i++) set.append(",'"+values[i]+"'");
     set.append(")");
   %>
   <!-- bug, luckily not mine but Java's setArray is not implemented so it has
   to be done by string concatenation -->
   <hermione:customEventList id="lel" userID="<%= userID %>" type="Lecture"
   listOfEvents="chosen2" >
     <sql:query>
     <%@ include file="chosen2.sql" %><%= set%>
     </sql:query>
     <sql:execute/>
   </hermione:customEventList>
   <hermione:filter type="add" >
     <hermione:source>evList</hermione:source>
     <hermione:source>chosen2</hermione:source>
   </hermione:filter>
   <%
     Vector uData= (Vector)pageContext.findAttribute("uData");
     Vector chosen2= (Vector)pageContext.findAttribute("chosen2");
     PropertyInterface user = (PropertyInterface)uData.elementAt(0);
     currentSemester = Integer.parseInt(user.get("currentSemester").toString());
   %>
   <sql:connection id="conn2">
    <sql:url>
      jdbc:microsoft:sqlserver://GORILLAZ:1433;Database=TryCourseScheduler;user=sunpr
      ess;password=sunpress;SelectMethod=cursor;
     </sql:url>
    <sql:driver>com.microsoft.jdbc.sqlserver.SQLServerDriver</sql:driver>
   </sql:connection>
   <sql:preparedStatement id="stmt4" conn="conn2">
    <sql:query>
      insert into isAssignedTo (StudentID, CourseEventID, duringSemester) values
      ('<sql:escapeSql><%= userID %></sql:escapeSql>',?,<%= currentSemester%> )
    </sql:query>
    <util:for varName="i" begin="0" iterations="<%=chosen2.size() %>">
     <sql:execute ignoreErrors="true">
      <sql:setColumn position="1">
        <%= ((PropertyInterface)chosen2.elementAt(i.intValue())).get("OID")%>
      </sql:setColumn>
     </sql:execute>
    </util:for>
   </sql:preparedStatement>
   <sql:closeConnection conn="conn2"/>
   <%response.sendRedirect("PortalAgenda.xjsp?appName=ChooseLectures");%>
   </util:If>
</ChooseLectures>
```

APPENDIX D

Extracts from the JavaDoc 1.3 generated documentation

Packages	
portal.taglib	
portal.taglib.advisor.ado	
portal.taglib.advisor.clustering	
portal.taglib.organizer	

Class Hierarchy

- class java.lang.Object
- class portal.taglib.organizer.**DateBean**
    - class portal.taglib.**DBconnPortalImpl**
    - class java.util.Dictionary
        - class java.util.Hashtable (implements java.lang.Cloneable, java.util.Map, java.io.Serializable)
            - class portal.taglib.organizer.**Event** (implements portal.taglib.organizer.PropertyInterface)
    - class portal.taglib.organizer.**Events**
        - class portal.taglib.organizer.**FreeEvents**
        - class portal.taglib.organizer.**LectureEvents**
    - class com.jacob.com.JacobObject
        - class com.jacob.com.Dispatch
            - class portal.taglib.advisor.ado.**Command**
            - class portal.taglib.advisor.ado.**Connection**
            - class portal.taglib.advisor.ado.**Field**
            - class portal.taglib.advisor.ado.**Fields**
            - class portal.taglib.advisor.ado.**Recordset**
    - class javax.servlet.jsp.tagext.TagExtraInfo
        - class portal.taglib.organizer.**CustomEventListTEI**
    - class javax.servlet.jsp.tagext.TagSupport (implements javax.servlet.jsp.tagext.IterationTag, java.io.Serializable)
        - class portal.taglib.advisor.clustering.**BaseQueryTag**
            - class portal.taglib.advisor.clustering.**GetColumnsTag**
            - class portal.taglib.advisor.clustering.**QueryTag**
        - class javax.servlet.jsp.tagext.BodyTagSupport (implements javax.servlet.jsp.tagext.BodyTag)
            - class portal.taglib.advisor.clustering.**CaseColumnTag**
            - class portal.taglib.advisor.clustering.**CaseTableTag**
            - class portal.taglib.advisor.clustering.**ConnectTag**
            - class portal.taglib.advisor.clustering.**CreateClusterTag**

            - class portal.taglib.advisor.clustering.**RetrainModelTag**
        - class portal.taglib.advisor.clustering.**DropModelTag**
        - class portal.taglib.organizer.**FilterTag**
        - class portal.taglib.advisor.clustering.**InputColumnTag**
        - class portal.taglib.advisor.clustering.**InputTableTag**
        - class portal.taglib.**ListTag**
        - class org.apache.taglibs.dbtags.preparedstatement.PreparedStatementImplTag (implements org.apache.taglibs.dbtags.statement.StatementTag)
            - class portal.taglib.organizer.**CustomEventListTag**
        - class portal.taglib.advisor.clustering.**RowsetDataTag**
        - class portal.taglib.organizer.**SourceTag**
            - class portal.taglib.organizer.**XMLSourceTag**
    - class portal.taglib.organizer.**EventListTag**
    - class portal.taglib.**IncludeApplicationTag**
    - class portal.taglib.organizer.**OutputXMLTag**
    - class portal.taglib.**UserGetParameterTag**
    - class portal.taglib.**UserSetParameterTag**
    - class portal.taglib.organizer.**WeekViewTag**
- class portal.taglib.organizer.**TodayBean**

Interface Hierarchy

- interface portal.taglib.advisor.ado.**CommandTypeEnum**
- interface portal.taglib.organizer.**PropertyInterface**

portal.taglib.advisor.ado
Interface CommandTypeEnum

```
public interface CommandTypeEnum
```

This interface stores all the possible values for the property `CommandType`. The `CommandType` property is used to set or return the type of a `Command` object.

Field Detail

adCmdUnspecified

```
public static final int adCmdUnspecified
```

This value indicates that the `CommandType` property has been unspecified.

adCmdUnknown

`public static final int` **`adCmdUnknown`**

This value indicates that the type of command in a `CommandText` property is not known. This is the default value.

adCmdText

`public static final int` **`adCmdText`**

This value evaluates the `CommandText` property as a textual definition of a command command or stored procedure call.

adCmdTable

`public static final int` **`adCmdTable`**

This value evaluates the CommandText property as a table name.

adCmdStoredProc

```
public static final int adCmdStoredProc
```

This value evaluates the CommandText property as a stored procedure.

adCmdFile

```
public static final int adCmdFile
```

Evaluates CommandText as the file name of a persistently stored Recordset. Used with Recordset. Open or Requery only.

adCmdTableDirect

```
public static final int adCmdTableDirect
```

Evaluates CommandText as a table name whose columns are all returned. Used with Recordset.Open or Requery only. To use the Seek method, the Recordset must be opened with adCmdTableDirect.

adExecuteNoRecords

```
public static final int adExecuteNoRecords
```

Indicates that the command text is a command or stored procedure that does not return rows (for example, a command that only inserts data). If any rows are retrieved, they are discarded and not returned. adExecuteNoRecords can only be passed as an optional parameter to the Command or Connection Execute method.

portal.taglib.advisor.clustering Class GetColumnsTag

```
java.lang.Object
  |
  +-javax.servlet.jsp.tagext.TagSupport
        |
        +-portal.taglib.advisor.clustering.BaseQueryTag
              |
              +-portal.taglib.advisor.clustering.GetColumnsTag
```

All Implemented Interfaces: javax.servlet.jsp.tagext.IterationTag, java.io.Serializable, javax.servlet.jsp.tagext.Tag

```
public class GetColumnsTag extends BaseQueryTag
```

JSP tag getColumns. This tag is used for creation of the new mining model with the specified model name and of the specified type. The default type of data mining is clustering. JSP Tag Lib Descriptor :

```
<name>create</name>
<tagclass>portal.taglib.advisor.clustering.CreateClusterTag</tagclass>
<bodycontent>JSP</bodycontent>
<info>A tag that creates new data mining model.</info>
<attribute>
  <name>model</name>
  <required>true</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
 <attribute>
  <name>queryRS</name>
  <required>true</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
```

```
<attribute>
  <name>attribute</name>
  <required>false</required>
  <rtexprvalue>true</rtexprvalue>
</attribute>
```

Fields inherited from class portal.taglib.advisor.clustering.BaseQueryTag
`caseIdColumn, caseTable, connectStr, inputIdColumn, inputTable, model, rowsetData, tableSource`

Constructor Summary
`GetColumnsTag()`

Methods inherited from class portal.taglib.advisor.clustering.BaseQueryTag
`getDistinctValues, getModelInfo, getRS, setModel`

Method Detail

setAttribute

`public void` **`setAttribute`**`(java.lang.String attribute)`

Sets the name of the property of a case that is requested.

Parameters:

`attribute` - the name of the requested case property.

setCaseID

`public void` **`setCaseID`**`(java.lang.String caseID)`

Sets the id for the case in question.

Parameters:

`caseID` - the identifier of the case.

setQueryRS

`public void` **`setQueryRS`**`(java.lang.String queryRS)`

Sets the name of the `Vector` object to store the requested data.

Parameters:

`queryRS` - the name of the `Vector` object.

portal.taglib.advisor.ado Class Recordset

```
java.lang.Object
  |
  +-com.jacob.com.JacobObject
        |
        +-com.jacob.com.Dispatch
              |
              +-portal.taglib.advisor.ado.Recordset
```

```
   public class Recordset extends com.jacob.com.Dispatch
```

Represents the entire set of records from a base table or the results of an executed command. At any time, the `Recordset` object refers to only a single record within the set as the current `record`.

Constructor Summary
`Recordset``()` Class constructor.
`Recordset``(com.jacob.com.Dispatch d)` This constructor is used instead of a case operation to turn a Dispatch object into a wider object - it must exist in every wrapper class whose instances may be returned from method calls wrapped in VT_DISPATCH Variants.

Method Detail

getBOF

```
public boolean getBOF()
```

BOF - Indicates that the current record position is before the first record in a Recordset object.

setCursorType

```
public void setCursorType(int pl)
```

There are four different cursor types defined in ADO:

- **Dynamic cursor (adOpenDynamic)** - allows viewing additions, changes, and deletions by other users; allows all types of movement through the Recordset that doesn't rely on bookmarks; and allows bookmarks if the provider supports them.
- **Keyset cursor (adOpenKeyset)** - behaves like a dynamic cursor, except that it prevents you from seeing records that other users add, and prevents access to records that other users delete. Data changes by other users will still be visible. It always supports bookmarks and therefore allows all types of movement through the Recordset.
- **Static cursor (adOpenStatic)**- provides a static copy of a set of records for you to use to find data or generate reports; always allows bookmarks and therefore allows all types of movement through the Recordset. Additions, changes, or deletions by other users will not be visible. This is the only type of cursor allowed when you open a client-side Recordset object.
- **Forward-only cursor (adOpenForwardOnly)**- allows you to only scroll forward through the Recordset.Additions, changes, or deletions by other users will not be visible. This improves performance in situations where you need to make only a single pass through a Recordset. Set the CursorType property prior to opening the Recordset to choose the cursor type, or pass a CursorType argument with the Open method.

Parameters:

```
pl - defines the type of the cursor. Possible values are:
           adOpenDynamic 2
           adOpenForwardOnly 0
           adOpenKeyset 1
           adOpenStatic 3
           adOpenUnspecified -1
```

getEOF

```
public boolean getEOF()
```

EOF - Indicates that the current record position is after the last record in a Recordset object.

getFields

```
public Fields getFields()
```

The `Fields` collection is the default member of the `Recordset` object.

Move

```
public void Move(int NumRecords, com.jacob.com.Variant Start)
```

It is possible to use the MoveFirst, MoveLast, MoveNext, and MovePrevious methods, the Move method, and the AbsolutePosition, AbsolutePage, and Filter properties to reposition the current record, assuming the provider supports the relevant functionality.

Open

```
public void Open(com.jacob.com.Variant Source,
com.jacob.com.Variant ActiveConnection, int CursorType, int LockType,
int Options)
```

Recordset objects can be created independently of a previously defined `Connection` object by passing a connection string with the Open method. ADO still creates a Connection object, but it doesn't assign that object to an object variable.

Update

```
public void Update(com.jacob.com.Variant Fields,
com.jacob.com.Variant Values)
```

`Recordset` objects can support two types of updating: immediate and batched. In immediate updating, all changes to data are written immediately to the underlying data source once you call the `Update` method.

UpdateBatch

```
public void UpdateBatch(int AffectRecords)
```

If a provider supports batch updating, it is possible to have the provider cache changes to more than one record and then transmit them in a single call to the database with the `UpdateBatch` method.

7 Literature

[ADO] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/his/thorref4_6ndu.asp

[AHL99] Leland Ahlbeck and Don Willits, *Pooling in the Microsoft Data Access Components,* Microsoft Corporation, May 1999, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmdac/html/pooling2.asp

[APACHE] http://www.apache.org

[ARGA95] R. Argawal, R. Srikant, *Mining sequential patterns,* Proceedings of the 11th International Conference on Data Engineering, pp.3-14, Zurich, Switzrland, September 1995

[ASP] http://www.aspin.com/

[BAL97] Marko Balabanovic, Yoav Shoham, *Fab : Content-based, Collaborative Recommendation*, Communications of the ACM, March 1997/Vol.40, No.3

[BAL97a] Marko Balabanovic, *An adaptive web page recommendation service.* In Proceedings of the 1st International Conference on Autonomous Agents, Marina del Rey, California, Feb ruary 1997

[CLAY01] Mark Claypool, Phong Le, Makoto Waseda, and David Brown, *Implicit Interest Indicators*, IUI'01, January 2001

[CH00] http://simulant.ethz.ch/Chariot/, 2000

[CHAN01] George Chang, Marcus J. Healey (Editor), James A. M. McHugh, Jason T. L. Wang, *Mining the World Wide Web - An Information Search Approach,* The Kluwer International Series on Information Retrieval, Volume 10) Kluwer Academic Publishers, June 2001

[CHUN00] Don Chun, *Using Portals to Enhance Your Self-Service Solutions,* Yvette Cameron, IHRIM Boston 2000 Spring Conference, June 2000

[CO02] http://cocoon.apache.org

[COOL99] R. Cooley, B. Mobasher, and J. Srivastava, *Data preparation for mining World Wide Web browsing patterns.* Journal of Knowledge and Information Systems, (1) 1, 1999

[DBTAG] *Jakarta Project: DBTags Tag library*, http://jakarta.apache.org/taglibs/doc/dbtags-doc

[DG00] The Delphi Group, *Portal Design Primer: 68 Questions for Portal Planners*, September 2000., http://www.delphigroup.com

[DOM] http://www.w3.org/DOM/

[ETH00] Conceptual Competition ETH World - Virtual and Physical Presence, 2000

[GAN00] John Ganci, Michael Adams,Mark Endei, Maria Miccolis, Erly Serrano, *WebSphere Personalization Solutions Guide*, IBM Redbook, December 2000

[GEAR01] David Geary, *Advanced JavaServer Pages*, Sun Microsystems Press, Prentice Hall Title, April 2001

[GG01] Gartner's Electronic Workplace & Intranets Research Note M-14-0730, http://www.gartner.com/webletter/sybase/october/article1.html, 23 July 2001

[GG02] G. Phifer, R. Valdes, D. Gootzit, Markets, Gartner Research Note, M-16-3524, 1 May 2002

[GLE01] Bernard W. Gleason, *Institutional Information Portal Key to Web Application Integration,* January 26, 2001

[GNAG01] Steven Gnagni, *PORTAL QUEST*, University Bussines Magazine, May 2001, http://www.universitybusiness.com/0105/feature.html

[GOLD92] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry, *Using collaborative filtering to weave an information tapestry*. Communications of the ACM, 35(12):61–70, December 1992

[GRIF00] Cary Griffith, *Vortals cut through Jabberwocky*, http://www.computeruser.com/articles/1904,5,15,1,0331,00.html, March 2000

[GRU87] J. Grudin, *Social Evaluation of the User Interface : Who Does the Work and Who Gets the Benefit?* Proceedings of IFIP INTERACT'87 : Human-Computer Interaction, 1987, pp.805-811

[HALL00] Marty Hall, *Custom Servlets and Java Server Pages*, Sun Microsystems Press, Prentice Hall Title, August 2000

[HAN00] Jiawei Han and Micheline Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, August 2000

[HAN97] E-H Han, G. Karypis,V. Kumar and B. Mobasher, *Clustering based on association rule hypergraphs*. In Proccedings of SIGMOD'97 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'97), May 1997

[HER01] Jonathn L. Herlocker, Joseph A. Konstan, *Content-Independent Task Focused Recommendation*, IEEE Internet Computing, pp. 40-47, November-December 2001, http://computer.org/internet

[HILL94] W.C. Hill, J.D. Hollan, *History-Enriched Digital Objects: Prototypes and Policy Issues,* The Information Society, 10 pp.139-145, 1994

[HUA98] Z. Huang, *Extensions to the k-means algorithm for clustering large data sets.* pp. 283-304, Data mining and Knowlegde Discovery(2), 1998

[INST00] Keith Instone, *Information Architecture And Personalization*, http://argus-acia.com/

[J2EE] Java 2 Platform, Enterprise Edition—Downloads and Specifications, http://java.sun.com/j2ee/download.html

[JACOB] Dan Adler, *The JACOB Project: A Java-COM Bridge*, September 2001, http://dandler.com/jacob/The JACOB Project.htm

[JAUS01] Kai Jauslin, *Framework for ETHWorld Portal*, March 2001

[JAXP] JAXP 1.1 JSR63, http://java.sun.com/aboutJava/communityprocess/review.html

[JaSIG] JA-SIG uPortal, http://mis105.mis.udel.edu/ja-sig/uportal/

[JNI] http://java.sun.com/products/jdk/1.2/docs/guide/jni/

[JOA97] Thorsten Joachims, Dayne Freitag, and Tom Mitchell, *WebWatcher : A Tour Guide for the World Wide Web*, Proceedings of the International Joint Conference in AI (IJCAI97), August 1997

[JSP] JavaServer Pages™ Specification Version 1.2 - Proposed Final Draft (PFD), October 26, 2000

[JSTL] JSP 1.2 Standard Tag Library (JSTL). http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html

[JSWP01] *Java Servlet Technology White Paper,* http://java.sun.com/products/servlet/whitepaper.html

[JTL] http://jakarta.apache.org/builds/jakarta-taglibs/nightly/projects/doc

[KAU97] Henry Kautz, Bart Selman and Mehul Shah, *ReferralWeb : Combining Social Networks and Collaborative Filtering*, pp. 63-65, Communications of the ACM, March 1997/Vol.40, No.3

[KOB01] Alfred Kobsa, *Generic User Modeling Systems, User Modeling and User-Adapted Interaction* 11, pp.49-63, Kluwer Academic Publishers, 2001

[KOE01] Alfred Kobsa, Juergen Koenemann, Wolfgang Pohl, *Personalized Hypermedia Presentation Techniques for Improving Online Customer Relationships*, The Knowledge Engineering Review 16(2), pp. 111-155 , Cambridge University Press, 2001

[KONS97] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl, *GroupLens : Applying Collaborative Filtering to Usenet news*, pp. 77-87, Communic ations of the ACM, March 1997/Vol.40, No.3

[KOUL99] Thomas M. Koulopoulos, *Corporate Portals: Make Knowledge Accessible to All*, http://www.informationweek.com/731/31erall.htm

[LDAP02] *OpenLDAP 2.1 Administrator's Guide : Intorduction to OpenLDAP Directory Services*, http://www.openldap.org/devewl/admin/intro.html

[LEV00] M. Levene, *The Navigation Problem in the World-Wide-Web*, 2000

[MCL00] Brett McLaughlin, Mike Loukides, *Java and XML*, O'Reilly & Associates; ISBN: 0596000162; June 2000

[MIL01] http://www.dla.mil/jgwi/definition_for_portal.htm

[MLA96] Dunja Mladenic, *Personal WebWatcher : Design and Implementation,* Technical Report IJS-DP-7472, Department of Intelligent Systems, J.Stefan Institute, Slovenia, 1996

[MOB00] Bamshad Mobasher, Robert Cooley, Jaideep Srivastava, *Automatic Personalization Based on Web Usage Mining,* Communications of the ACM, August 2000/Vol.43, No.8, pp.142-151

[MOR99] David Morrison, Martin Buckley and Steve Cappo, *Building a Portal with Lotus Domino R5,* IBM Red Book, October 1999

[MUL00] Maurice D. Mulvenna, Sarabjot S. Anand, Alex G. Buechner, *Personalization on the Net using Web Mining*, pp.123-125, Communications of the ACM, August 2000/Vol.43, No.8

[NIEL98] Jakob Nielsen, *Personalization is Over-Rated,* Alertbox for Oct. 1998, Personalization, http://www.useit.com/alertbox/981004.html

[OARD97] Douglas W. Oard, *The State of the Art in Text Filtering. User Modeling and User-Adapted Interaction*, pp. 141-178., http://www.glue.umd.edu/~oard/ research.html

[OARD98] Douglas W. Oard and Jinmook Kim, *Implicit Feedback for Recommender Systems*, Proceedings of AAAI Workshop on Recommender Systems, July 1998

[OCONN99] Mark Connor and Jon Herlocker, *Clustering items for collaborative filtering.* ACM SIGIR '99

[OLEDB] OLE DB for Data Mining Specification, Version 1.0, Microsoft Corporation, July 2000

[PMML] http://www.dmg.org/pmmlspecs_v2/pmml_v2_0.html

[PAZZ96] M. Pazzani, J. Muramatsu and D. Billsus, *Syskill & Webert: Identifying interesting web sites.* In Proccedings of the 13 th National Conference on Artificial Intelligence, 1996

[PORT80] M. Porter, *An algorithm for suffix stripping* Program 14(3) pp.130-137, 1980

[PRAC] http://www.practicalportals.com

[RAS02] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K. Lam, Sean M. McNee, Joseph A. Konstan, John Riedl, *Getting to Know You : Learning New User Prefrences in Recommender Systems,* IUI'02, San Francisco, California, January 2002

[RES97] Paul Resnick and Hal R. Varian, *Recommender Systems*, pp.56-58 Communic ations of the ACM, March 1997/Vol.40, No.3

[REG] http://jakarta.apache.org/regexp/

[RFC] D. Kristol, L.Montulli, *HTTP State Managenment Mechanism*, http://www.ietf.org/rfc.rfc2109.txt

[ROC71] J. Jr. Rocchio, *Relevance feedback in information retrieval.* In The Smart System Experiments in Automatic Document Processing pp.313-323, Prentice Hall Inc, 1971

[RUCK97] James Ruckner and Marcos Polanco, *Siteseer : Personalized Navigation for the Web*, pp.73-75, Communications of the ACM, March 1997/Vol.40, No.3

[SAP01] SAP White Paper: *mySAP technology Portal Infrastructure : People-Centric Collaboration*, 2001

[SAX] SAX 2.0 Specification, http://www.megginson.com/SAX/

[SAL83] G. Salton and M. J. McGill, *An Introduction to Modern Information Retrieval*. McGraw-Hill, 1983

[SAR01] Badrul Sarwar, George Karypis, Joseph Konstan and John Reidl, *Item-Based Collaborative Filtering Recommendation*, WWW10, May 2001

[SCHA01] Thomas Schaeck, IBM Software Group, *WebSphere Portal Server and Web Services Whitepaper*, 2001

[SCHAR95] Upendra Schardanand, Pattie Maes, *Social Information Filtering: Algorithms for Automating 'Word of Mouth'*, CHI95 Proceedings

[WDAV] http://www.webdav.org/

[XALAN] http://xml.apache.org/xalan-j/overview.html

[XERCES] http://xml.apache.org/xalan-j/overview.html

[XSLT] XSLT Specification, http://www.w3c.org/XSLT

Contents

Index of figures