Construction of Adaptive Web-Applications from Reusable Components

Guntram Graef and Martin Gaedke

Telecooperation Office (TecO), University of Karlsruhe, Vincenz-Priessnitz Str. 1,
76131 Karlsruhe, Germany, Tel.: +49 (721) 6902-79, Fax: -16,
E-Mail: {graef|gaedke}@teco.edu

Abstract. The Web has become a ubiquitous environment for application delivery. The originally intended idea, as a distributed system for knowledge-interchange, has given way to organizations offering their products and services using the Web as a global point of sale. The centralized delivery-mechanism enables the construction of E-Commerce applications personalized for each user by using behavior analysis. Current technologies suffer from the Web's legacy and use Log file-analysis or collaborative filtering only to adapt the content to users' needs. Motivated by the results of collaborative filtering algorithms, we describe a construction approach based on the abstract concept of services. To support the fine-grained concept we use the component-based WebComposition Markup Language to support reuse and seamless evolution of E-Commerce applications.

1 Introduction

While originally intended as a medium for distributing information in a document-centric form, the World Wide Web [3] has become much more than that. With the introduction of dynamism both on the client-side and the server-side the Web has emerged as a new platform for software applications serving a variety of purposes such as E-Commerce.

An important aspect that distinguishes applications on the Web from PC or work station applications is their central deployment. Software serving a large number of users can be installed and maintained in one single location. This drastically reduces cost and duration needed for deploying new or multiple versions of an application. It also means that the behavior of users can be observed in a single location.

Maes and Shardanand demonstrated in [15] how centrally acquired data about user interests can be utilized to provide users with information adapted to their preferences. With the work presented in this paper we go one step further. Rather than dynamically generating single html pages with information such as product recommendations we adapt complete E-Commerce applications to the needs of individual customers.

A fundamental requirement for the automatic adaptation of applications in a flexible and evolution oriented way is the availability of all necessary functionality

and features as independent building blocks or components. The document based implementation model of the Web neither allows for the granularity needed for this kind of components nor does it provide mechanisms for abstraction. Thus it is for example not possible to implement a component as a normal Web resource that represents a feature such as a corporate identity design applicable to all Web applications of an organization. We can overcome this problem with WebComposition, a component-based approach to Web development that we will detail at the beginning of third section of this contribution. After that we will introduce services as higher level functional abstractions and describe how we use a service factory and domain-specific languages to improve productivity and maintainability for service development. We will conclude that section with our mechanism for the automatic adaptation of an application to individual customers' needs. In the fourth section we will detail the algorithm we employ for identifying individual customers' requirements. An evaluation of our work and a conclusion can be found at the end of this contribution. Beforehand, in the following section, we will discuss the current state of art concerning individual and adaptive applications.

2 Previous work

The problem of directly addressing a user and his or her personal needs has been subject to research for many years. Much of this work has been centered around user interface development, which is quite natural since this is the part of an application most visible to users. There is also some work related to adaptive Web sites.

Solutions can be divided into adaptive or individual (parts of) applications. An adaptive application presents a general solution that adapts its behavior during run-time to the user. An individual application in contrast has been produced for the specific needs of a user.

2.1 Adaptive and Individual User Interfaces

In the user interface design community there has been work on both adaptive user interfaces as well as the development of individual user interfaces. Adaptive user interfaces are widely discussed in contributions like FLEXEL [17] or [14], but their adaptation process is limited to the analysis of a single user's behavior. Individual user interface development has been addressed with TADEUS [13] and ADEPT [11]. These tools include a user model, which is used to automatically produce individual user interfaces or user interface descriptions. A major limitation is that the user model has to be defined manually by the UI developer.

2.2 Adaptive Web Sites

In the Web community there has been work on adaptive Web sites following several different approaches. Projects such as WebWatcher [2]or AVANTI [5] use a *path prediction* approach. This technique is closely tied to the structure of hypertext consisting of information resources (documents) and links. Path prediction tries to forecast the next action a user wants to perform after reaching a given state. The notion of state is usually tied to the recently viewed document resource. In a more complex E-Commerce application, state is not limited to currently viewed resources but might also include such things as state of product configurations created by a customer in advance of an ordering process which might make path prediction much more complex. Both examples also rely on manual input from the user about his personal interests.

Another technique is *collaborative filtering*. It is based on user preferences and user behavior in the past rather than navigation between application states. This approach described in [15] uses information about user interests to determine groups of users with similar interests. The calculated results can be used as recommendation of resources for a user based on previously accessed resources by other users in the same group. The idea has been demonstrated in an experimental implementation called Ringo. About 2000 users entered ratings for audio-cd-titles and music artists. This data has been used to dynamically generate html-pages with music recommendations for each user.

While Ringo requires users to actively provide information about their preferences, some commercial applications such as Amazon.com [1] try to infer the likes and dislikes of its customers from observed actions such as ordering a product. This proved to be a feature crucial for user acceptance of any type of adaptive mechanism in E-Commerce applications. In a highly competitive environment such as the Internet it is mandatory to avoid or minimize any customer workload.

3 Evolution and Adaptation of E-Commerce Applications

The automatic adaptation of whole applications on the level of code primitives is not feasible with current technology. We need to find proper abstractions and components as application building blocks to reduce the complexity of the application adaptation process. Therefore we divide the construction of software into a supply oriented micro level for providing functional components and a demand oriented macro level where functional components are federated in the application according to user demand.

The only kind of code abstraction ubiquitous on the Web are document resources. Unfortunately these do not satisfy our requirements both in terms of supported granularity and expressive power of inter-document relations. To overcome these limitations we based our approach on the WebComposition programming model.

On top of this general programming model we use the notion of services as higher level building blocks encapsulating functionality required to perform a certain task such as placement of an order or providing customer feedback. In contrast to lower level system components, the function of services can be perceived and described without specific technical knowledge. Service components form the basis for macro level application adaptation and evolution.

To support the development and seamless evolution of a large number of services we introduce a process for service production based on a domain specific language and a service factory. The macro level adaptation of E-Commerce applications is done through a special application service. This service controls how other services are presented to the customer based on the automatic analysis of customer requirements detailed later in this contribution.

3.1 WebComposition

The WebComposition programming model is based on the WebComposition Markup Language [8] and the WebComposition approach [10]. It enables us to perform component-based software development for the Web in a platform independent and evolution oriented way.

3.2 Reuse oriented programming model

A major productivity factor during all cycles of software development is reuse. The WebComposition programming model aims at facilitating reuse of development artifacts of any kind of granularity on various levels of abstraction. Code fragments of any given target language are modeled as WebComposition components. Components define a first level of code abstraction.

Between WebComposition components it is possible to define reuse relations such as aggregation (has-part) and specialization (inherits-from). Thus the model is object-oriented. More specifically it is based on a *prototype-instance paradigm* [18] instead of a class-based object-oriented model. This means that instantiated objects can inherit the capabilities of components by simply using them as prototypes. It is not necessary to provide class definitions. In the WebComposition model an object is an instance of a component and a component can serve as a prototype for other components. A component may be used like an abstract class, i.e. it could serve as a prototype for a certain type of components.

Prototyping as described in [18] is a mechanism to implement code sharing among objects. Alternatively multiple references of a component could be used to share its code. Sharing is fundamental for efficient reuse and maintenance because it enables us to keep modifications local. This is in contrast to other suggested object-oriented Web models such as WOOM [4].

3.3 WebComposition Markup Language

Components are described using the WebComposition Markup Language (WCML). WCML extends the semantics of common Web languages with statements for component definition and inter-component relations. WCML is an application of the eXtended Markup Language (XML). This way we can build on the familiarity of Web developers with markup languages and the availability of various tools such as syntax parsers. Components are organized in *virtual component stores,* which are implemented as documents, database tables or Web server resources. A *component repository* allows for retrieval of components through various access models [7]. WCML components can be mapped automatically to the Web implementation model using a compiler. This mechanism is independent of any specific deployment platform or language as the programming model is generic. Migration to a new implementation language or simultaneous support of several implementation languages can be achieved by using object-oriented concepts like polymorphism and WCML components encapsulating target language specific code.

3.4 Services as Functional Abstractions

The typical function of a Web based E-Commerce system is to enable customers to perform tasks such as product ordering or information retrieval. In a more general sense we can abstract from the concrete possibilities a system offers to its customers. Instead we can state that an E-Commerce system provides a set of services to its users. Each product, news channel or item of information can be modeled as a service provided by the system and the organization behind it. As a concrete example we could realize the selling of office chairs through such a service.

This definition of a service provides us with a powerful task oriented abstraction. A service as a whole tends to change very little when technical changes occur as compared to less abstract components.

Services are based on a level of abstraction both familiar to customers as well as domain experts in an organization. Interest in and knowledge about the requirements of a certain service is often found in a single person or small organizational unit. Customer activities most easily can be described as performing a set of tasks. Implementing tasks as service components we can base our application adaptation process on customer access to and federation of a set of components. Changes in an application on a task-level granularity also does not tend to bring the same level of confusion sometimes found when changes occur on a sub-task level. Services usually consist of components describing different aspects such as layout, navigation, language, content and processing. A more detailed description of service components can be found in [9].

3.5 Automated Service Production

Due to our practical experience with E-Commerce application environments there tend to be groups of services that only differ from each other in rather limited ways but can account for a large percentage of the total amount of services needed. For the development of this kind of services we try to bridge the gap between less technically sophisticated domain experts and the development system by means of introducing a service factory.

Our service factory uses service descriptions to automatically produce components implementing a service in the WebComposition programming model. A service description contains all information necessary to distinguish a specific service from a general type of services. Thus a service for ordering office chairs might be of the general type of a product order service while possessing numerous specific properties describing its products, configuration options, user interface dialog elements and so on. The service description can be specified using a special visual editor or a simplified human readable markup language. In contrast to many code production tools available, maintenance of the services remains on the service description level.

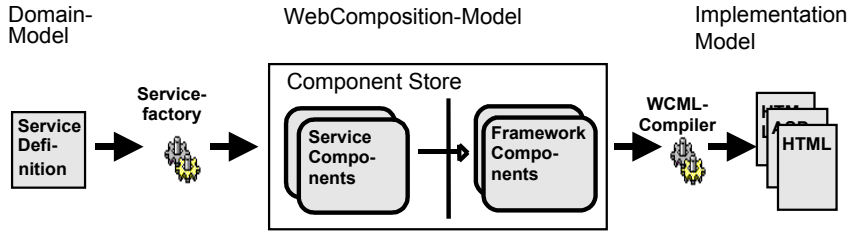

Figure 1: Service production

The service components produced by the service factory inherit from existing WebComposition components that form a component framework for service development. Thus basic changes affecting a larger group of services such as all information services can be done by changing a single set of components in the framework. The service descriptions for single services stay unaffected. This separation of concerns into different components and levels of the system architecture allows for an incremental evolution of individual services and groups of services [6]. The process of service production has been summarized in figure 1.

3.6 Adaptation Process

We consider a Web application as a special kind of service that contains all the functionality a system offers to its users. In most cases it will contain other services tailored to more specific tasks. We call this service the application service.

The adaptation process (figure 2) is performed by the application service. It is based on customer requirements data. In the next section we will therefore describe how this data can be obtained and processed automatically. The application service

combines other services and makes them available according to their importance to the customer. Thus a service available in the overall system might either be presented to a customer more or less prominently or hidden. In this way the customer's cognitive capacity and the available bandwidth for system to customer interaction can be used in an optimized way.

4 Individual Customer Requirements Analysis

Determining the needs of an individual customer is the basis for providing an adaptation mechanism for any kind of Web application. A customer needs certain functionality presented to him or her in a way suiting his or her preferred style of interaction. In this section, we will discuss the selection of functionality in the form of services in more detail.

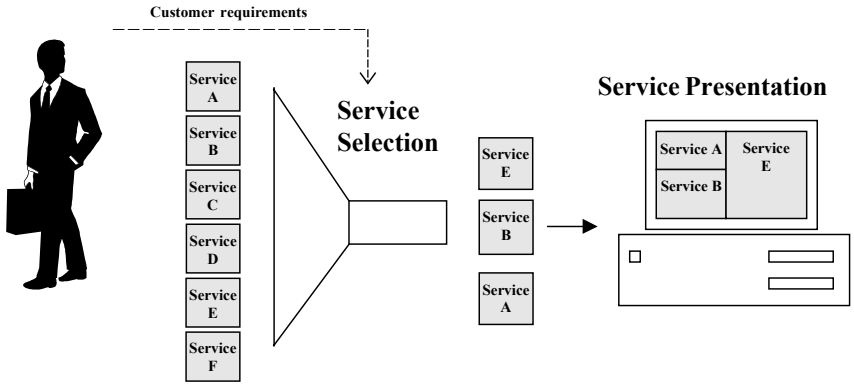

Figure 2: Adaption to customer requirements

4.1 The Service Selection Problem

We have a given set of services S, a set of customers C and a set of customer actions A in the past obtained from the observation of customer behavior. Each action $a \in A$ is related to a service $s \in S$ and a customer $c \in C$.

The problem is how to find a set of Services S(c) for a customer c that most accurately reflects those services the customer would use in the future assuming equal exposure of each service to the customer.

A simple approach to this problem is to calculate S(c) directly from the set of services related to past actions of the same customer c. In single user environments this usually is the only possible solution. We will use this method to determine those services that are regularly used by a customer. The algorithm implementing this method will be referred to as the *conservative service selection algorithm*.

Obviously not all services a customer might be willing to use in the future can be derived from the customer's past actions alone. Fortunately in our multi-customer case we have access to multiple customers' activity data. Thus we can employ a second algorithm for determining services of use to a customer, which we call the *social service selection algorithm*.

4.2 Observation of Past Customer Behavior

Before we can apply any algorithm for service selection we need to obtain A, the set of past customer actions. A very widespread approach to gathering information about user activity on the Internet is through logging HTTP-requests made to the Web server. This works if each customer action that is of interest is directly related to a set of URLs on the Web server. It also assumes that the result of the action (such as success of subsequent processing on the server) as well as the current application state is not relevant. In our E-Commerce scenarios both assumptions couldn't be made, due to the use of dynamic URLs and complex application state. Thus we where forced to explicitly log certain user actions from within our application when they occurred.

4.3 Conservative Service Selection Algorithm

We call this algorithm "conservative", because all services selected have already been used by the same customer before and no new suggestions are added. First, for each customer it is determined which services $S_{used} \subset S$ were used in the past. The number of successful invocations of a service by the customer, as indicated by the customers past actions A_c, is counted. A service s has been successfully used if e.g. an order has been completed or an information item has been viewed long enough. Finally for each customer c the n most heavily used services from S_{used} are selected as the result set S(c).

4.4 Social Service Selection Algorithm

Our approach for this algorithm is based on the implicit exploitation of recurrent patterns in user behavior. Behavioral patterns state that people that exhibited a certain past behavior B_p are likely to exert a certain behavior in the future B_f with recent and expected future behavior forming a complete behavioral pattern [16]. A simple such pattern might be that a person that once used a service to order a telephone installation will very likely (with probability p) use another service to buy accessories for that telephone.

One way to exploit this for service selection is the explicit definition of association rules [12] capturing these patterns. This requires explicit and up to date knowledge of the existing behavioral patterns, which is usually not available because their number and complexity grow exponentially with the number of Services. Even for the most trivial case where each behavior is only related to one service, the number of possible relations between past user behaviors and predicted user behaviors is $O(|S|^2)$.

Our approach is based on the implicit exploitation of behavioral patterns. Instead of directly associating observed actions via rules with services, we do so indirectly by deriving associations between customers. The algorithm we suggest consists of the following steps:

1. *Customer Profile Generation*: For each customer $c \in C$ generate a profile p based on all actions $a \in A$ related to that customer. The set of customer profiles is called P.
2. *Determination of User Profile Similarity Relation*: Calculate a similarity relation between each pair of customer profiles in P.
3. *Target Profile Generation*: For each $p \in P$ calculate a target profile t.
4. *Service Selection*: Use the target profile t to determine the service selection S(c).

A customer profile consists of an attribute vector. Each attribute describes the level of usage of a service by the customer. We applied the following mapping of customer activities to profile attributes:

0 = No past actions by the customer related to that service
1 = Service was used by the customer at least once, but never successfully completed
N = Service successfully was used by the customer N-1 times.

To determine the degree of similarity between two customer profiles we applied a distance metric based on the mean squared difference between two profile vectors:

$$\frac{1}{\overline{(P_x - P_y)^2}}$$

We use the set of profiles and the similarity metric to generate a "target profile" for each user. This profile tries to predict what a customer's profile will develop to if equally exposed to all available services. It is based on the assumption that people who behaved similar in the past will do so in the future. It also assumes that if a customer C_1 has only partially completed a pattern of behavior, another customer C_2 whose profile shows a high degree of similarity compared to C_1's profile might already have executed the remaining part of that pattern. The optimal way to compute the target profile would be the calculation of a weighted average from all available customer profile vectors, where the weight would be a function of the similarity calculated in the previous step of the algorithm. However, for performance reasons we decided to calculate the target profile as the average vector from the n profiles that are most similar to the considered customer's profile but that are significantly different in at least one attribute. This is to avoid the effect of profile convergence and stabilization if there are many identical profiles (e.g. initial profiles of new customers).

After generating the target profile t we compute a differential profile d = t - p. We then use the relations between profile attributes and customer actions and between customer actions and services to determine the service selection S(c). Our result set contains services that if used by the customer would change his profile to become more similar to the target profile.

4.5 Quality of Predictions

We conducted a simple evaluation about the quality of predictions made by the different algorithms. As the basis for the algorithms we took a set of history data about past activities of about 1000 users of the Eurovictor II system described in the next section, and compared our predictions to activities performed by the same users afterwards. The complete set of existing services was equally exposed to each user. Thus every user was subject to the same application experience without any adaptation done during this stage.

We compared the number of selected services that have actually been used during the following two weeks while applying four different selection algorithms: Random, Conservative Service Selection Algorithm (CSSA), Social Service Selection Algorithm (SSSA) and a combination of both CSSA and SSSA, where each of the two algorithms contributed to the result set.

Applied algorithm	Predicted services used per customer
Random	0.7
CSSA	4.5
SSSA	6.2
CSSA and SSSA combined	6.8

Figure 3: Quality of predictions

Figure 3 shows the results. Please note that the numbers in the table should only be interpreted relative to each other, because the absolute values also depend on the actual type and number of services and the present behavioral patterns. Obviously, both CSSA and SSSA produce useful predictions, while the combination of the two algorithms delivers better results than the application of a single algorithm.

4.6 Runtime Complexity

The conservative service selection algorithm's complexity is relatively low. Complexity for one customer scales linear with the number of services in the system, being O(|S|).

Social service selection proves much more complex, since for the selection of services for each customer, data from all other customers has to be considered. Thus the algorithms complexity for one customer basically is O(|C| * |S|).

5 Evaluation

The approach presented in this paper has been used to implement a complete service oriented E-Commerce system. Eurovictor II has been developed in the Telecooperation Office at the University of Karlsruhe in a joint project with Hewlett-Packard (HP). The system is currently productive at HP in Europe with more than

10000 regular customers. Eurovictor II has been designed as an evolvable and open system providing services to internal customers at HP.

The system already offers several hundred services mainly in the areas hardware orders, software orders, telecommunication, virtual office and news services. Services have been developed in a decentralized manner by people with moderate technical skills and have automatically been integrated into the system. Eurovictor II is highly flexible and adapts itself to each customer based on past activities of all users. It provides each of its users with an individual application experience.

Figure 4 shows a screenshot from Eurovictor II as it has automatically adapted itself to a specific customer. The application contains Eurovictor II services that have been selected using a combination of the two service selection algorithms described in the previous section. The presentation of the services has also been adapted to the customer's habits. Frequently used services are displayed more prominently or even executed automatically. Right after the start of the Web application the "application changes"-service is automatically invoked, because it has been accessed by the current customer more frequently than other services. This specific service retrieves and displays information about the most recent changes during the evolution of what the customer perceives as personal Eurovictor II application. Another customer might be welcomed by Eurovictor II with a service that displays a stock report or a company news summary.

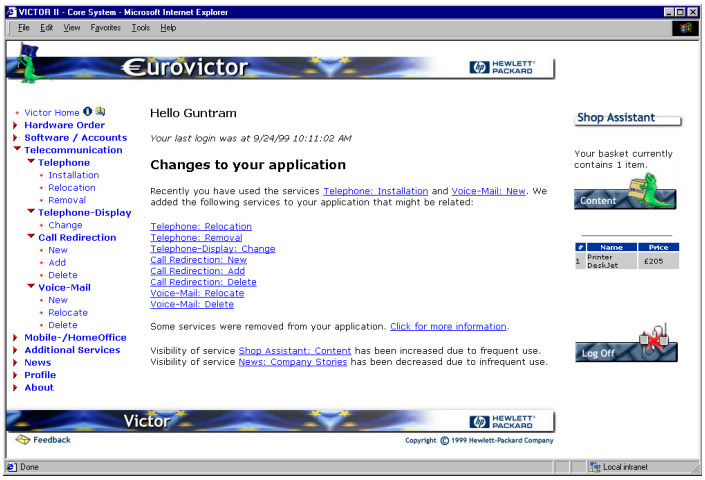

Figure 4: Adaptive Eurovictor II

6 Conclusion

In this paper we have described a platform based on the so-called “service” abstraction. The platform uses a component-based development model for the Web. Based on this model we introduced a service factory that enhances productivity

during Web service development and allows technically less sophisticated domain experts to enable business processes on the Web. An application service together with a mechanism for the automatic analysis of individual customer requirements enables the automatic adaptation of applications to individual customers' needs. Components implementing solutions to different problems are used and reused as building blocks.

From the individual customer's point of view the application performs an automatic evolution, based on the changing behavior of all customers. The evolution of services due to changes in business logic is separated from this process and centralized. Logic closely related to technology is encapsulated in separate components. Thus, we achieve a clear separation of concerns related to three major sources of change and fields of expertise: customer requirements, technology and business processes.

As a proof of concept we successfully applied our approach in a mission critical E-Commerce environment. The results from Eurovictor II encourage us to look at further issues such as adaptation on a finer granularity optimizing the way individual tasks can be performed by the customer. Another issue would be the detection and consideration of a customer's level of user expertise in addition to the requirements in terms of system functionality.

References

[1] AMAZON.COM, Amazon Homepage: http://www.amazon.com (accessed: May 2000)

[2] R. ARMSTRONG, D. FREITAG, T. JOACHIMS, T. MITCHELL, WebWatcher: a learning apprentice for the World Wide Web, in: AAAI Spring Symposium, Stanford, U.S., pp. 6-12.

[3] T. BERNERS-LEE, Information Management: A Proposal: CERN. 1998. http://www.w3.org/Proposal.html

[4] F. CODA, C. GHEZZI, G. VIGNA, F. GARZOTTO, Towards a Software Engineering Approach to Web Site Development, in: 9th International Workshop on Software Specification and Design (IWSSD), Ise-shima, Japan.

[5] J. FINK, A. KOBSA, A. NILL, User-oriented adaptivity and adaptability in the AVANTI project: Microsoft Usability Group, Redmond, Washington, USA 1999. http://fit.gmd.de/hci/projects/avanti/publications/ms96.html

[6] M. GAEDKE, H.-W. GELLERSEN, A. SCHMIDT, U. STEGEMÜLLER, W. KURR, Object-oriented Web Engineering for Large-scale Web Service Management, in: Thirty-Second Annual Hawaii International Conference On System Sciences (HICSS-32), Island of Maui, USA.

[7] M. GAEDKE, J. REHSE, G. GRAEF, A Repository to facilitate Reuse in Component-Based Web Engineering, in: International Workshop on Web Engineering at the 8th International World-Wide Web Conference (WWW8), Toronto, Ontario, Canada.

[8] M. GAEDKE, D. SCHEMPF, H.-W. GELLERSEN, WCML: An enabling technology for the reuse in object-oriented Web Engineering, in: Poster-Proceedings of the 8th International World Wide Web Conference (WWW8), Toronto, Ontario, Canada.

[9] M. GAEDKE, K. TUROWSKI, Generic Web-Based Federation of Business Application Systems for E-Commerce Applications, in: Second International Workshop on Engineering Federated Information Systems (EFIS'99), eds. S. Conrad, W. Hasselbring, G. Saake, Kühlungsborn, Germany.

[10] H.-W. GELLERSEN, R. WICKE, M. GAEDKE, WebCompostion: an object-oriented support system for the Web engineering lifecycle, Computer Networks and ISDN Systems Special Issue on the 6th Intl. WWW Conference, Santa Clara, CA, USA 1997 1429-1437.

[11] P. JOHNSON, S. WILSON, P. MARKOPOULOS, J. PYCOCK, ADEPT: Advanced Design Environment for Prototyping with Task Models, in: Human factors in computing systems (CHI'93), Amsterdam, The Netherlands, pp. 56.
[12] R. MILLER, Y. YANG, Association Rules over Interval Data, in: ACM SIGMOD international conference on Management of data, Tucson, Arizona, USA, pp. 452-461.
[13] E. SCHLUNGBAUM, T. ELWERT, TADEUS - a model-based approach to the development of Interactive Software Systems, Rostocker Inform. Berichte 17 1995 93-104.
[14] M. SCHNEIDER-HUFSCHMIDT, T. KÜHME, U. MALINOWSKI, Adaptive user interfaces : principles and practice, Amsterdam; New York, 1993.
[15] U. SHARDANAND, P. MAES, Social information filtering: algorithms for automating "word of mouth", in: Human factors in computing systems (CHI'95), Denver, USA, 210-217
[16] B. F. SKINNER, Science and human behavior, New York, 1953.
[17] C. THOMAS, M. KROGSÆTER, An adaptive environment for the user interface of Excel, in: international workshop on Intelligent User Interfaces (IUI), pp. 123-130.
[18] D. UNGAR, R. B. SMITH, Self: The Power of Simplicity, in: OOPSLA '87, pp. 227-242.