An Agent Architecture For Personalized Web Stores

L. Ardissono, C. Barbero, A. Goy, G. Petrone
Dip. di Informatica, University of Torino
Cso Svizzera 185; 10149 Torino, Italy
Phone: +39 - 011 - 7429111;
E-mail: {liliana, cris, goy, giovanna}@di.unito.it

ABSTRACT

We describe the architecture of a configurable Web store supporting personalized interactions with users. Our system exploits user modeling and flexible hypermedia techniques to tailor to the user the suggestion of goods and the description of the store catalog. Customizing the system's behavior requires the parallel execution of several complex tasks during the interaction (e.g. identifying the user's preferences and dynamically generating the hypertextual pages of the store catalog). We argue that only an architecture composed of specialized agents can successfully carry on these tasks.
Our system is developed in a Java-based environment using tools for building interoperable, agent-based systems (e.g. JDBC drivers for database independence; Voyager for the multiagent architecture). The system architecture includes an instance of each specialized agent and an agent which enables the communication among the others.

1. INTRODUCTION

In this paper, we describe a tool to build adaptive Web stores which tailor the selection and the presentation of products to the specific needs of their customers.

With the recent expansion of the Internet, the interest towards electronic sales has quickly grown and many tools have been built to help vendors to set up their Web stores: for example, Oracle's Internet Commerce Server, Microsoft Merchant Server, NetConsult Communications' Intershop Online, Netscape Merchant [22]. These tools offer all the facilities necessary to build the databases of the store and to manage the order processing and secure payment transactions. However, they typically do not focus on issues like the personalization of the interaction with the customers. On the other hand, Web surfers are generally heterogeneous and have different needs and preferences. For this reason, personalizing the interaction with the user and the product presentation is becoming more and more important.

Personalization issues are crucial to improve the usability of Web sites, to provide the users with descriptions tailored to their domain expertise, to select the information which best matches their interests, and even to offer them different interaction techniques (e.g. interfaces adaptable to people with special needs [12]). The design of adaptive Web systems has however highlighted the fact that complex architectures have to be defined, where several specialized components cooperate with each other to obtain the overall system behavior; see for instance [24, 18].

In particular, the customization of a Web store differs with respect to the situation of a generic recommender system for many aspects; most noticeably:

- The search space is represented by the store catalog, which is a local information source. So, the description of the goods sold in the store can be structured and detailed, and complex forms of reasoning can be performed to evaluate their properties and their suitability to the user's preferences.
- The user accesses the store with the specific goal to purchase goods and the system should assist her in the selection of the products which best match her needs. On the other hand, the adoption of information filtering techniques suited to deal with huge and heterogeneous information sources (as those used by the information filtering systems which navigate the Web to retrieve items for the user [2]) is most suited to the satisfaction of generic user goals. Therefore, it may not be the most appropriate means to actively help the user in the selection of goods.

For these reasons, we believe that a knowledge-intensive approach better achieves the goal of building flexible and friendly Web shops.

Our analysis of the electronic sales domain, as far as the customer-to-business side is concerned, has revealed the importance of several tasks, among which:

- Managing the various phases of the interaction with the customer (e.g. user identification and catalog browsing). In each phase, the system has to decide whether it needs to ask information to the user, which actions the user is allowed to perform, and so on. Moreover, it has to monitor the user's actions, in order to collect information which can then be used to adapt the system's interaction style to the user's specific needs.
- Deciding how to adapt the system's interaction style to the user: the contents (and links) of the hypertextual pages have to be tailored to the user's characteristics and interests, depending on her behavior [12]. For example, if she follows links in order to get more technical information about the products displayed, the system may then autonomously provide such information in the rest of the interaction; if, in contrast, the user asks for help (e.g. because she does not understand the terminology used), then the system may continue the interaction by describing products using simpler words. Moreover, more or less detailed pages may be designed depending on the user's receptivity.

- Dynamically generating the hypertextual pages which represent the Web store: only few pages can be precompiled, while the others have to be generated on the fly, depending on the user's short-term goals (e.g. which products she wants to see and which information about the products she requests).

- Acquiring and maintaining the user profiles, which represent the system's hypotheses on the users' needs and can be used to tailor the layout of the hypertextual pages to each specific user.

- Managing the customers database to keep their previous purchases and their profiles available later on.

- Managing the database of the items sold in the store.

- Tailoring the suggestion of goods to the user's tastes, by evaluating which items best match her preferences.

These activities are instances of very different issues, which require different types of expertise (knowledge and inferences) in order to be solved; for example, a description may be customized by applying personalization strategies [9], while the acquisition of the user profile could rely on techniques like the use of stereotypical information about customer classes, or the application of dynamic user modeling rules [26, 17].

The presence of heterogeneous problems to be faced and the fact that many tasks could be carried on in parallel while interacting with the user suggests that a monolithic architecture, composed of a single agent, cannot manage the overall problem in a satisfactory way (see also [24, 15]). For this reason, we have decided to develop several specialized components, each of them carrying on one of the main tasks outlined above. In order to allow a harmonic cooperation among the different components, we designed our system as a multiagent architecture, where agents can be distributed on a network. Each agent offers a specific set of services and knows which ones are offered by the other agents of the system. We have mainly focused on two problems:

- The identification of the subproblems which the generic "personalized sales" task could be decomposed into.

- The identification of the domain-dependent / independent knowledge required to solve such subproblems: in our system, the domain-dependent knowledge, which concerns information about products and customer features, is declaratively represented and clearly separated from the domain-independent components, which represent the core of the store. This separation has the advantage that our architecture can be easily instantiated on several sales domains, therefore obtaining different Web stores out of a single Web store shell.[1]

In the following sections we present the architecture of our system: we will refer to the telecommunication domain (e.g. selling faxes, switchboards, etc.), which is the application domain selected for our prototype; however, the system is designed to accommodate other commercial domains, where services, or complex products, are massively sold.

The Web stores created by our system interact with the user in a cooperative way; they handle a personalized interaction with the user, offering different views of the catalog on the basis of her needs and preferences. The system can take the initiative to suggest certain products, which may differ from those explicitly requested by the user, because, according to the system's knowledge, they better suit her needs. Each Web store dynamically generates the personalized hypertextual pages containing the information to be shown to the user and the links she can follow next. In such pages, the system displays information about the products and lets the user perform actions, like exploring other products, getting information about a specific product and selecting a good to purchase.

[1]In order to make the store customization task easier, we have designed graphical acquisition tools which help the store designer to introduce the domain-dependent knowledge.

2. THE ARCHITECTURE

Our Web store shell has been designed as the multiagent architecture shown in Figure 1. In the figure, boxes represent agents; ovals represent the Users and Products databases, which keep specific information about the customers who visited the store and the items sold there. The parallelograms represent the dynamic structures maintained during each working session: the user profile (User Model - UM) and the knowledge bases used by the agents, i.e. the Stereotype KB, which describes the characteristics of customer classes, and the Product Taxonomy, which is a conceptual representation of the products available in the store.
The dashed lines relate the knowledge bases to the data structures initialized on the basis of their contents: for instance, at each interaction the user profiles are generated and initialized on the basis of a template, defined within the Stereotype KB. Similarly, the Product Taxonomy View, which represents the personalized hyperspace, is generated out of the conceptual representation of products represented in the Product Taxonomy.
The dotted arrows show the flow of the data retrieved and stored by the agents during their activity: e.g. the User Modeling Component (UMC) retrieves the description of the stereotypes from the Stereotype KB; moreover, it cooperates with the Users DB Manager to read and store in the Users DB the information about the specific customers who visited the store.
The arrows between agents show the messages exchanged by them during a working session: the thin arrows denote asynchronous messages, while the thick ones denote synchronous messages. Each arrow has associated the message type sent by the agent; we have labeled the message types with names derived from the KQML performatives [11].

In the following, we briefly describe the various agents, focusing on the services they offer and providing some examples of the messages they exchange. More information about the activities of the agents can be found in [4, 5].

2.1 Session Manager

The Session Manager handles the communication with the browsers and creates all the agents of the architecture. Each time a user connects to the store, the Session Manager sends a synchronous "tell" message to the Dialog Manager, to communicate the user's action (e.g. she wants to follow an hypertextual link or she has filled a form). We will describe the Session Manager in more detail in Section 3.

2.2 Dialog Manager

The Dialog Manager handles the interaction with the user and maintains the contextual information. The interaction

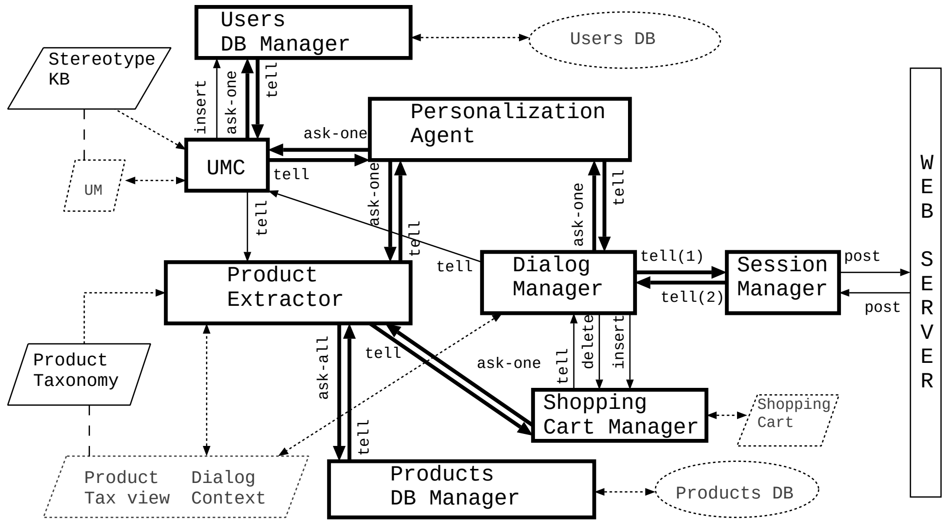


Figure 1: The Web store architecture: rectangles denote agents; parallelograms denote knowledge bases. Bold (thin) arrows represent synchronous (asynchronous) messages between agents.

is composed of three main phases:

1. During the identification phase, new customers are identified and their personal data are collected.
2. During the selection-browsing phase, the user may cycle selecting products she wants to examine and browsing the catalog:
   (a) Selection: the user may choose one or more products she wants to analyze. The system keeps track of her selections to describe their features later on. The user is also allowed to select products for other beneficiaries than her.
   (b) Browsing: the user browses the portion of the virtual catalog describing the products previously selected. During this phase, the user can see the description of the main product features and of those related to the specific items included in the Products DB. Moreover, the user can select some items to purchase, putting them into the shopping cart.
3. The user may close the interaction: when she presses the "Exit" button, the system asks her if she allows her data to be permanently stored in the Users DB and, depending on her answer, it records or discards them.

The Dialog Manager monitors the user's actions, by interpreting the generic events caught by the Session Manager. We have identified a set of events relevant to dynamic user modeling purposes; when these events occur, the Dialog Manager sends the information to the UMC, which exploits it to update the user profile.
The Dialog Manager also guides the user's browsing activity within the Product Taxonomy View. On the basis of the link followed by the user, the Dialog Manager asks the Personalization Agent for a new page and the interaction continues along the same line.

2.3 User Modeling Component

The UMC creates and maintains the profiles of the customers of the on-line store.[2] It also triggers the construction of the Product Taxonomy View by (asynchronously) invoking the Product Extractor, when the contents of the user profile change ("tell" message): in this way, the view can be updated during the session.
At the end of a session, if the user agrees, the UMC makes the Users DB Manager store the user profile into the Users DB (by means of an asynchronous "insert" message).

Each user profile contains:

- Descriptive information about the user; this information concerns personal data, like the user's age, education level and job, which are provided by the user by filling a registration form. This information is represented as <feature, value> pairs.
- Predictive information, concerning the user's preferences towards the properties of goods, like quality and ease of use, and the user's personality traits (e.g. her receptivity).
  Each user preference is represented by means of a parameter, structured as follows:[3]

  Parameter Name: String;
  Importance: Integer [0..1];
  Values: <Linguistic Value, Probability> pairs;

  The Importance slot represents how much the property is considered important by the user and takes values in the range [0..1].
  Each <Linguistic Value, Probability> pair contains a

[2]For each user session, the UMC keeps a profile for the direct user and one for each beneficiary of the selected products (e.g. see Figure 3, which shows a products presentation page relative to a user session where the user is searching for a phone for a beneficiary named Paul). The direct user profile is used to tailor the presentation of items; the other profiles are used to select the goods suited to the person they are directed to.

[3]We leave apart the representation of personality traits, which is very similar to that of preferences (but has no importance slot). This representation is derived from [25].

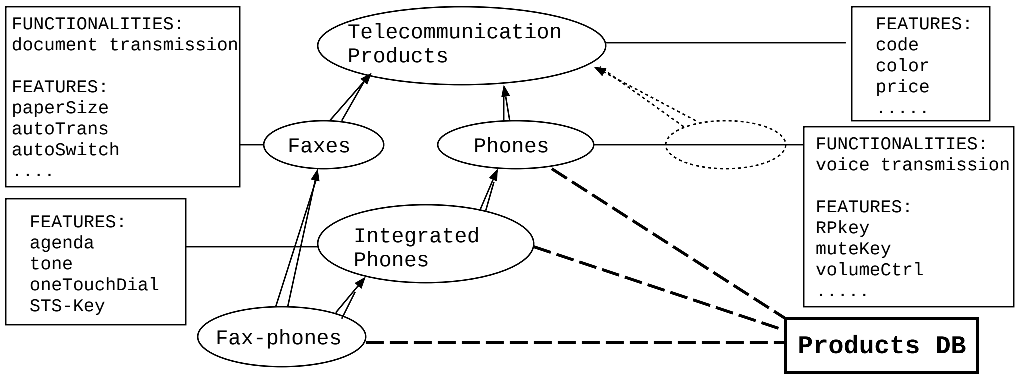


Figure 2: A portion of the Product Taxonomy. The double arrows represent generalization links. The solid lines relate products to their features. The dashed lines are links to the Products DB.

linguistic value that the parameter can assume and the probability that the user prefers that linguistic value for the related product property. For instance, the user's preference towards the products quality might be instantiated as follows:
<Low, 0>, <Medium, 0.3>, <High, 0.7>.

When a user accesses the store, her user profile is initialized either by retrieving the user's record from the User's DB (by means of a synchronous "ask-one" message to the Users DB Manager, which replies with a "tell" message), or by exploiting stereotypical information about customers, available in the Stereotype Knowledge Base.

The Stereotype KB contains a hierarchical taxonomy of stereotypes which cluster typical properties of customer groups [21]. The stereotypes are characterized by a classification part and a predictive part. The classification part concerns socio-demographic characteristics, corresponding to the personal data asked to the user when she first accesses the Web store. The predictive part concerns personality traits and user preferences towards product properties. The user is classified by matching her personal data with the classification part of the stereotypes. Then, the user's personality traits and preferences are initialized by merging the predictions of the best matching stereotypes. The contribution of each stereotype is weighted on the basis of the degree of match between the user's personal data and the classification part of the stereotype. In this way, strictly matching stereotypes influence the initialization of the user profile more heavily.

Recent developments of our system include the exploitation of dynamic user modeling techniques to update the user profiles on the basis of the actions the user performs during the interaction. The UMC maintains in its working memory the history of all the relevant events delivered by the Dialog Manager. The events may regard clicks to hypertextual links and clicks on buttons to perform specific tasks (e.g. putting an item into the shopping cart, creating a comparative table to compare items on the basis of a set of features). The system periodically analyzes the event history to evaluate how frequently the user performs the various actions. On the basis of these data, the user's characteristics are updated by the system, so that the interaction style may be modified.[4]

[4]For instance, if the user often requests more information about products (by clicking on the "more features" links, see Figure 3), her receptivity is upgraded and, as a consequence, the system will provide the user with more information in the subsequent steps of the interaction.

Moreover, the set of items contained in the shopping cart is exploited to update the user's preferences so that they reflect the properties of the chosen items.

2.4 Product Extractor

The Product Extractor Agent dynamically builds a Product Taxonomy View to offer the user a personalized, constrained view on the complete static hyperspace represented by the Product Taxonomy. This view is used by the Dialog Manager to guide the user in a rational, assisted exploration of the catalog, instead of directly accessing the item descriptions in the Products DB. Figure 2 shows a portion of the Product Taxonomy built for our prototype and shows the product attributes within rectangles.

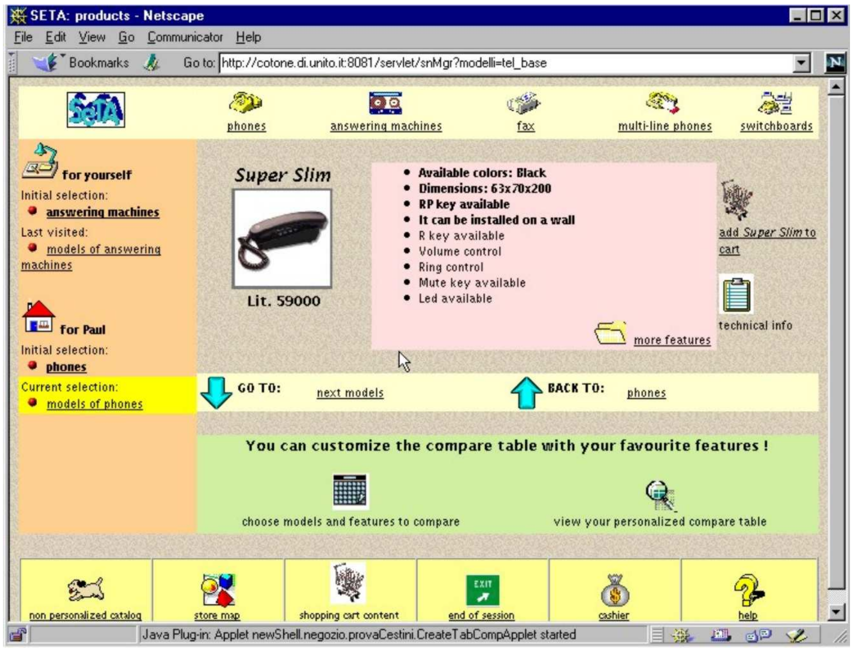
The Product Extractor also assists the user's selection of the products she wants to see: it checks the Product Taxonomy View, looking for possible integrated solutions which satisfy her needs; for instance, if the user asks for a phone and a fax, the system proposes a "fax-phone" as an integrated solution. Note that this behavior displays a system initiative, since the system reacts to the user's requests showing products which serve her needs better than the ones she explicitly asks for.

At last, the Product Extractor retrieves from the Products DB Manager the internal description of all the items which might be suggested to the user (by means of a synchronous "ask-all" message to the Products DB Manager, which replies with a "tell" message) and selects the ones which best match the beneficiary's preferences, by comparing the properties of the retrieved items with the preferences of the beneficiary taken from the UMC.

2.5 Personalization Agent

The Personalization Agent dynamically generates the HTML code for the hypertextual pages which it is required to produce. Although this agent only works when invoked by the Dialog Manager, it has a certain autonomy of action: in fact, the Dialog Manager only specifies the type of page to be produced and which product must be described; on the basis of such data, the Personalization Agent takes all the decisions regarding the layout and the contents of the page. After having produced the page, this agent sends it to the Dialog Manager by means of a synchronous "tell" message.

In the page generation process, the Personalization Agent requests the data of the user profile from the UMC; then it



- Available colors: Black
- Dimensions: 63x70x200
- RP key available
- It can be installed on a wall
- R key available
- Volume control
- Ring control
- Mute key available
- Led available

You can customize the compare table with your favourite features !

Figure 3: A product presentation page dynamically generated by our system.

applies a set of personalization rules which, on the basis of the user profile, suggest the page layout and the amount and surface form of the page contents. Therefore, different pages are produced when describing the same product to users characterized by different profiles.

Figure 3 shows a page produced by the Personalization Agent, describing the "Super Slim" phone: while the central area of the page describes the features of this item, the lefthand side area keeps a memory of the user's previous selections and the upper part allows the user to inspect the main product classes described in the catalog (it can be used in order to select new products to examine). The rest of the page contains links and buttons which allow the user to perform actions, like adding goods to the shopping cart and browsing the catalog.

2.6 Shopping Cart Manager

The Shopping Cart Manager keeps track of the items selected by the user to purchase during an interaction. To this purpose, it stores the chosen items into a structured shopping cart, keeping track of the various beneficiaries and uses (home vs. business use).

When the user picks up a good to purchase, the Dialog Manager sends to the Shopping Cart Manager an asynchronous, one-way message ("insert"), reporting the selected item, its beneficiary and use. At each time during the interaction, the user can consult the state of the shopping cart (in order to display such information, the Personalization Agent asks the list of items to the Shopping Cart Manager): the goods in the shopping cart are displayed, together with their beneficiary, use and price. If the user removes a good from the shopping cart, the Dialog Manager sends to the Shopping Cart Manager an asynchronous, one-way message ("delete").

2.7 DB Managers

The database managers (Products DB Manager, Users DB Manager) handle the Products and Users databases, which contain, respectively, the information about the goods sold in the store (e.g. price, technical features, etc.) and the records of the customers who have accessed the store, including the goods that they purchased in previous visits.

3. IMPLEMENTATION DETAILS

Our system is implemented in Java as a Three Tier Application Architecture (see Figure 4).

- The First Tier can use any Java-enabled browser;[5]
- The Second Tier contains the Web Server and most of the agents of our system. The interface between the agents and the Web is supported by a Servlet (the Session Manager). Our store is based on the JavaSoft Java Web Server 1.1, which we initially chose because it was the only Web Server supporting the Servlet API.[6]
- The Third Tier consists of data repositories and can be accessed using relational database interfaces, like JDBC. Both the Products and Users DB Managers use JDBC to communicate to the DBMS: JDBC is the Java API that allows the developer to write a single interface to the most popular relational databases.[7]

[5]The complex user interface tasks, like the initial selection of the product types the user is interested in, are handled by Java applets downloaded from the second Tier servers; the simpler tasks are handled using standard HTML, dynamically generated by the Personalization Agent.

[6]Currently, also the Netscape, Microsoft and Apaches Servers support Servlets.

[7]In our system, the independence from the structure of the database is enhanced by the existence of the Product Taxonomy: in fact, this structure represents an interface which allows the system's components to refer to the product descriptions using a rich, concep-

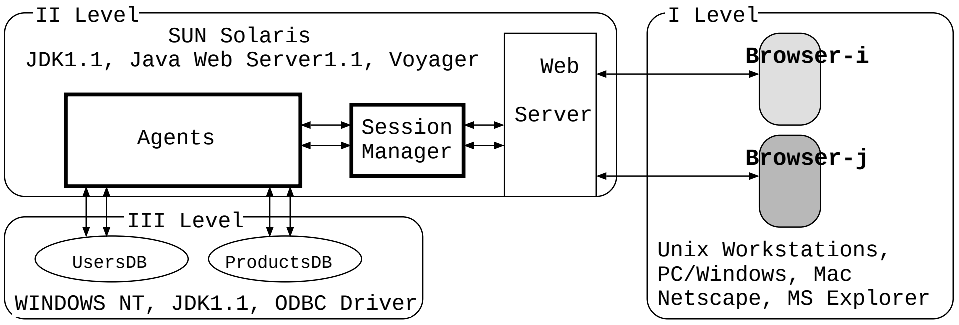


Figure 4: Our Three Tiers Application Architecture.

In our current prototype, the Users and Products databases (Third Tier) are stored on a NT machine with the Microsoft Access DBMS, while the Second Tier of our system runs on a Solaris workstation. In principle, however, most of the agents could be distributed over different computers.

We have implemented the system architecture by exploiting Objectspace Voyager [19], a tool for building multiagent systems which provides system components with the basic agent capabilities (e.g. distribution and communication protocols). In particular, the agents of our system are Java objects which offer all the services necessary to carry on the personalized interactions with customers; moreover, we have exploited Voyager's facilities to allow the agents' distribution over different computers and their communication.

The only exception is the Session Manager, which is implemented as a Servlet. Servlets are used to extend Web server capabilities; in fact, they have been designed to build Web based systems. In particular, we have exploited them for two main tasks:

- To handle the communication with the browsers (catching users' actions and sending to the users' browsers the Web pages of the store).
- To conveniently handle concurrent user sessions (by exploiting session tracking) in our Session Manager.

The agents of the system exist during the whole life of the virtual store as unique instances of Voyager objects. However, since many users may connect to the virtual store, such agents must serve multiple user sessions in parallel; they do this as follows:

- At the store initialization time, the Session Manager creates all the agents of the system (UMC, Product Extractor, etc.) as Voyager objects. These objects contain the user independent data and code, which are maintained in a single copy, so that the knowledge bases, like the Product Taxonomy, are not replicated for each user session.
- When a user starts a new session, the Session Manager assigns a unique "userId" to it; later on, the Session Manager catches the user's actions and delivers the messages to the agents, specifying the appropriate userId. For each user session, each agent creates a new object in order to handle the session-dependent data. Moreover, each agent maintains a private table where it stores the references to those objects, which are retrieved by the associated userIds.

Figure 5 sketches the parallel sessions handled by the UMC; the other agents work in the same way, but the figure does not show their sessions for simplicity.

The presence of agents which handle multiple user sessions might lead to think that they represents bottlenecks within the architecture. However, the multithread environment supported by Voyager overcomes this problem. In fact, while synchronous messages are handled by the main thread of an agent, the agents are spawned when they receive asynchronous messages which, therefore, are handled in parallel. In this way, we can handle in a homogeneous way an almost complete parallelization of the activity of the various agents within a user session (see the asynchronous messages in Figure 1) as well as different sessions.

3.1 Comments

We have studied several tools for building multiagent systems in a Java-based environment: all these tools offer communication and distribution facilities, and introduce an abstraction level with respect to the communication protocol (which might be RMI, DCOM, CORBA, or other). We have selected Objectspace Voyager [19], which best suited the needs of our architecture, allowing a convenient object distribution: objects created by the Voyager compiler are able to be remotely executed. Moreover, Voyager supports an almost seamless transformation of a Java object, which can only exchange synchronous messages, into an agent able to send and receive various types of messages (noticeably, although different message synchronization types are supported, the Java method declaration does not change). Voyager offers synchronous, oneWay, oneWayMulticast and "future" messages ("future" messages correspond to asynchronous messages, which do not involve the sender and the receiver in a rendez-vous). Messages are exchanged among agents by method invocation; however, the object invoking a method has to specify in which way it wants to synchronize with the receiver. While synchronous communication corresponds to traditional Java method invocation, the other types of messages can be delivered by objects by invoking specific Voyager methods.

Other systems, like JAFMAS [10] and JATLite [1], would allow our agents to be distributed on a network and communicate in a standard agent communication language based on Speech Acts [23, 11, 20]. However, following KQML, they impose that the contents of the speech acts are represented

tual representation, and abstracting from the specific commitments made by the designers of the database; such a representation allows relating the product features with the specific representation of items in the database.

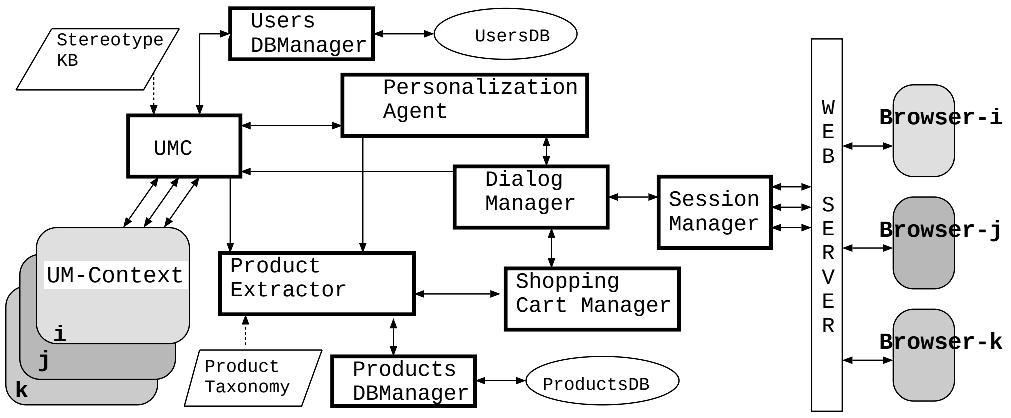


Figure 5: Parallel user sessions in the Web store. The arrows denote the flow of data between agents and objects in general.

(in an appropriate content language) as strings. Therefore, they make the exchange of messages containing structured information a complex task.[8] On the other hand, in the development of our system we have experienced that a huge portion of the information which the agents need to send to each other is embedded into complex objects and a conversion of such data into very general String patterns would make the communication among the agents inefficient.

Since we need flexibility in agent communication, we excluded other well known agent building tools, which are explicitly focused on the management of agent mobility, at the expense of agent communication, that is handled at a rather low level: e.g. MOLE [8], and IBM Aglets [13]. At the same time, other tools for building multiagent systems which also provided primitives to enable an active cooperation between the agents proved to exceed our needs: e.g. COOL [6]. In fact, in our system, the agents offer fixed services, and there is no need to dynamically distribute tasks among them.

4. CONCLUSIONS

We have described a configurable Web store architecture which can be used to build stores offering personalized interactions to customers, on the basis of their domain expertise, preferences and needs. In order to achieve the personalization of the interactions with the user, we have applied several techniques, developed within the Knowledge Based Systems, Flexible Hypermedia and User Modeling research areas, and adapted them to the characteristics of the electronic sales application.

We have designed our Web store shell as a multiagent system, in order to allow the creation and coordination of several, specialized agents, who are responsible for carrying on the activities necessary to the management of the personalized interactions with customers. In a Web store instance, these agents can be distributed over different computers and interact with each other by exploiting the communication facilities offered by the Voyager tool to build agent-based systems.

Following the described architecture, we have built a prototype store shell and we have instantiated it on the telecommunication domain: the resulting system can be used to browse the catalog of a store selling phones, faxes, and other similar products. No real purchasing can be performed, but the hypertextual pages of the catalog are generated by the system by consulting a real database of goods, which contains all the information about the items available in the store.
During the interaction with a customer, the system builds a user profile by classifying the user in stereotypical customer classes, on the basis of a set of socio-demographic parameters (which we selected for the telecommunication domain). The system dynamically generates the hypertextual pages describing the catalog, tailoring the product presentation and the hypertextual links to the user's preferences. Moreover, the system tailors the selection of items to the user's characteristics: within a product class selected by the user, the system recommends the items which best match the preferences of the beneficiary to which they are directed.

In the design of our system, we have adopted a knowledge-intensive approach: this fact requires that, when a new store instance is created, detailed information has to be introduced to describe the product features, as well as the main characteristics and preferences of customers (stereotypical information). Moreover, the system builds and exploits rather complex user profiles in order to select the products to suggest and to tailor their description to the user's needs. Clearly, we could adopt such an approach because the information sources used by the system to process the user's requests are well-structured and constrained. On the contrary, other recommender systems, which search for information spread all over the Web, must face the heterogeneity of the information sources; therefore, they use simpler customer profiles and more generic matching techniques to identify items of interest to the user (e.g. [2, 16, 14, 3, 7]).

Future work includes a field trial of the system and the resolution of several issues, among which:

- The enhancement of the system's initiative during the interaction with the users: e.g. more personalization techniques will be added, among which a help facility for non-expert users.
- The enhancement of the system's configurability: we are currently separating the different capabilities of the system in order to support the selection of the functionalities required within a store instance. For example, the User Modeling Component can currently work in static user modeling, stereotypical user modeling and dynamic user modeling modality.[9]

[8]Object serialization could be used to transmit messages containing object parameters, but a sensible effort is required to decode and parse the content of messages on the receiver's side.

[9]The dynamic user modeling activity is currently only partially

We are also working to improve the dynamic generation of the textual descriptions produced by the system, in order to offer a (easy to configure) multilingual system.

- The investigation of the interoperability of our system: in fact, the exploitation of tools like Voyager warrants the communication platforms (CORBA, etc.) interoperability. However, our system currently has no capability to register and advertise services with matchmakers and brokers, for example, to allow a broker to collect information about the goods available in the store on behalf of a shopper agent.

5. ACKNOWLEDGMENTS

This work is developed in the project "Servizi Telematici Adattativi" (http://www.di.unito.it/˜ seta), carried on at the Dipartimento di Informatica of the University of Torino within the national initiative "Cantieri Multimediali", granted by Telecom Italia. We are grateful to L. Console, L. Lesmo, C. Simone and P. Torasso for having contributed significantly to this work with suggestions and fruitful discussions.

REFERENCES

[1] JATLite. http://java.stanford.edu/java_agent/html.

[2] Recommender systems. COmmun. ACM, 40, 3, 1997.

[3] Ackerman, M., Billsus, D., Gaffney, S., Hettich, S., Khoo, G., Kim, D.J., Klefstad, R., Lowe, C., Ludeman, A., Muramatzu, J., Omori, K., Pazzani, M.J., Semler, D., Starr, B., and Yap, P. Learning probabilistic user profiles. AI Magazine (Summer 1997), 47–55.

[4] Ardissono, L., Goy, A., Meo, R., and Petrone, G. An agent architecture for personalized interaction with customers in virtual stores. In Proceedings Int. IFIP/GI Working Conf. on Trends in Distributed Systems for Electronic Commerce TrEC'98 (Hamburg, 1998), Dpunkt Verlag, Heidelberg, 137–148.

[5] Ardissono, L., Goy, A., Meo, R., Petrone, G., Console, L., Lesmo, L., Simone, C., and Torasso, P. A configurable system for the construction of adaptive virtual stores. World Wide Web, to appear.

[6] Barbuceanu, M., and Fox, M.S. The design of a coordination language for multi-agent systems. In Muller, J.P., Wooldridge, M.J., and Jennings, N.R. (eds.), Lecture Notes in Artificial Intelligence n. 1193 Intelligent Agents III. Springer, 1996.

[7] Barra, M., Cattaneo, G., Izzo, M., Negro, A., and Scarano, V. Symmetric adaptive customer modeling for electronic commerce in a distributed environment. In Lamersdorf, W., and Merz, M. (eds.), Lecture Notes in Computer Science n. 1402 Trends in Distributed Systems for Electronic Commerce. Springer, 1998, 11–25.

[8] Baumann, J., Hohl, F., Radouniklis, N., Rothermel, K., and Straβer, M. Communication concepts for mobile agent systems. In Proceedings 1st Int. Work. on Mobile Agents MA'97 (1997).

integrated into the system, which monitors the user's activity but does not change the user profiles runtime.

[9] Brusilovsky, P. Methods and techniques of adaptive hypermedia. User Modeling and User-Adapted Interaction, 5, 2-3 (1996), 87–129.

[10] Chauhan, D. JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation. PhD thesis, University of Cincinnati, Stanford CA, 1997.

[11] Finin, T.W., Labrou, Y., and Mayfield, J. KQML as an agent communication language. In Bradshaw, J. (ed.). Software Agents. MIT Press, Cambridge, 1995.

[12] Fink, J., Kobsa, A., and Nill, A. Adaptable and adaptive information access for all users, including disabled and the elderly. In Proceedings 6th Conf. on User Modeling (Chia, 1997), 171–173.

[13] IBM. Aglets. http://www.trl.ibm.co.jp/aglets/whitepaper.htm.

[14] Bellcore Inc. Adapt/x recommender system. http://www.bellcore.com/ADAPTX/index.html.

[15] Jennings, N.R., Sycara, K.P., and Wooldridge, M. A roadmap of agent research and development. In Proceedings of Autonomous Agents and Multi-agent Systems (Boston, 1998), Kluwer Academic Publishers, 275–306.

[16] Krulwich, B. Lifestyle finder - intelligent user profiling using large-scale demographic data. AI Magazine, 18, 2 (1997), 37–45.

[17] McTear, M.F. User modelling for adaptive computer systems: a survey of recent developments. Artificial Intelligence Review, 7 (1993), 157–184.

[18] Milosavljevic, M., and Oberlander, J. Dynamic hypertext catalogues: Helping users to help themselves. In Proceedings the 9th ACM Conference on Hypertext and Hypermedia HT'98 (Pittsburgh, 1998).

[19] Objetcspace. Voyager. http://www.objectspace.com/index.asp.

[20] Petrie, C.J. Agent-based engineering, the web, and intelligence. IEEE Expert (December 1996), 24–29.

[21] Rich, E. Stereotypes and user modeling. In Kobsa, A., and Wahlster, W. (eds.), User Models in Dialog Systems. Springer, Berlin, 1989, 35–51.

[22] Seachrist, D. Hanging out Internet shingle. BYTE (April 1997), 136–139.

[23] Steiner, D. An overview of FIPA 97. http://drogo.cselt.stet.it/fipa/#Papers.

[24] Sycara, K.P., Pannu, A., Williamson, M., and Zeng, D. Distributed intelligent agents. IEEE Expert (December 1996), 36–45.

[25] Torasso, P. and Console, L. Diagnostic Problem Solving. North Oxford Academic, 1989.

[26] Wahlster, W., and Kobsa, A. User Models in Dialog Systems. Springer, Berlin, 1989.