A software architecture for dynamically generated adaptive Web stores[*]

Liliana Ardissono, Anna Goy, Giovanna Petrone and Marino Segnan
Dipartimento di Informatica - Università di Torino
Corso Svizzera 185, Torino, Italy
{liliana,goy,giovanna,marino@di.unito.it}

Abstract

We provide technical details about the software and hardware architecture of SETA, a prototype toolkit for the creation of Web stores which personalize the interaction with customers. SETA is based on a multi-agent architecture and on the use of MAS technologies, which support a seamless communication among the system agents and an easy distribution of such agents on different computers.

1 Introduction

Personalization has recently become a central focus of attention for Web-based systems [Riecken, 2000]. This paper describes the software and hardware architecture of SETA [Ardissono *et al.*, 1999; 2000], a prototype toolkit for the creation of adaptive Web stores, which tailor the suggestion and the presentation of products to the individual customer's needs. We will provide details about the SETA multi-agent architecture, the class hierarchy which defines the internal architecture of the system agents and the technologies used to support agent distribution, communication and specific agent activities. The main prototype store created using our system presents telecommunication products, like phones and faxes.

Sections 2 and 3 sketch the application domain and the adaptivity functionalities offered by the Web store. Section 4 describes multi-agent architecture. Sections 5 and 6 describe the management of multi-user access and communication among system agents. Sections 7 and 8 specify the internal design of a SETA agent and the external software used to obtain specific system functionalities. Section 9 provides technical details and section 10 compares our approach to the one of other commercial Web-based systems. Section 11 closes the paper.

[*]This paper describes the evolution of a system developed at the CS Department of the University of Torino, within the project "Servizi Telematici Adattativi", funded by Telecom Italia. We thank L. Console, L. Lesmo, C. Simone and P. Torasso for their comments to this work.

2 Application domain

We designed our system to satisfy personalization requirements in the Business to Customer area of e-commerce, focusing on the presentation of massively sold, medium-complexity products, such as home appliances. In this area, personalizing the presentation of goods is important because the customer has to select items out of a rich pool of alternatives. This decision depends on factors ranging from the price of the items to their features and the customer needs a lot of information to choose the item suiting her needs at best. Moreover, depending on her interests, not all the product features might be equally important: thus, the presentation can be dramatically improved by focusing on the most relevant information. Another reason for personalization is that the customer might not be aware of all the functionalities offered by a product class, therefore needing to be assisted in the identification of the relevant goods. Thus, the system should both personalize the presentation of the catalog and elicit information about the user's needs, in order to actively suggest alternative products.

Although e-commerce has strong adaptivity demands, it constraints the development of user interfaces in several ways. For instance, Web stores must be accessible via standard equipments, such as a personal computer, a (usually slow) Internet connection and a Web browser. Moreover, the time needed to browse the catalog and find the needed products must be as short as possible and the run-time efficiency of the system is essential. Finally, Web store shells, such as SETA, must satisfy the requirements of the store designer, possibly reducing the overhead in the configuration of a new Web store.

3 Adaptivity in SETA

A Web store catalog generated by SETA is organized as a hypertext containing two main page types:

(1) Pages presenting product classes in a synthetic style, specifying the main functionalities offered by the related items: e.g., the description of the faxes product class specifies that faxes transmit documents and some of them also make photocopies. These pages also provide the user with the hypertextual links for navigating the catalog.
(2) Pages showing detailed information about the items of a product class. These pages present the features offered by the individual items and provide the user with rich interaction functionalities: e.g., they enable her to ask for technical details, to create comparison tables "on the fly" and to put items into the shopping cart.

SETA dynamically generates the pages of a Web store catalog by exploiting personalization strategies for selecting the information to be presented on the basis of the user's interests and familiarity with the products. Moreover, the system presents the available items for a product class (e.g., the available fax models) sorting them on the basis of the user's preferences [Ardissono and Goy, 2000]. During the interaction, the system monitors the user's selections to identify her needs for product functionalities and suggest potentially interesting product classes which the user has not visited. In this way, the assistance is extended to the search for alternative products.

4 Architecture of SETA

The management of personalized interactions results from the coordination of several activities like user modeling, dynamic page generation, etc., requiring the exploitation of different knowledge and techniques; e.g., specific strategies are needed for tailoring layout, content and structure of the pages to the users. To address such complexity, modular architectures, integrating components devoted to the different tasks, are applied in the design of adaptive systems on the Web. The exploitation of agent-based technologies is even more effective, as MAS technologies support the development of distributed systems where heterogeneous agents offer specialized services and interact to produce the overall service in an open environment; e.g., [Jennings *et al.*, 1998; Giampapa *et al.*, 2000; Macho *et al.*, 2000].

In the development of our system, we have exploited knowledge representation techniques and agent-based technologies to improve the configurability of the toolkit and its scalability. SETA is based on a multi-agent architecture where specialized agents fill the main roles for the management of personalized interactions with customers; e.g., user modeling and generation of Web pages [Ardissono *et al.*, 1999]. Each agent handles multiple user sessions and maintains session-specific data in parallel session environments. In the following we sketch the most relevant agents.
At the Web store starting time, the **Session Manager** creates the SETA agents and provides them with the references to the other SETA agents they might need to send messages to. During the life-span of the Web store, the Session Manager handles the communication with the browsers, catching the user's actions and forwarding them to the Dialog Manager.
The **Dialog Manager** monitors the user's actions and maintains an interaction context storing information about the navigation in the catalog; after each user action, this agent choses the page to be produced next, asks the Personalization Agent to generate it and forwards the page to the Session Manager, to send it to the browser.
The **User Modeling Component (UMC)** maintains the models of customers, revising them during the interaction, on the basis of their behavior.
The **Product Extractor** supports the personalized suggestion of goods by ranking items on the basis of how strictly they match the user's preferences.
The **Personalization Agent** dynamically generates the code for the catalog pages by exploiting the information about the user provided by the UMC and a set of personalization rules for selecting information about products and type of description to be produced [Ardissono and Goy, 2000].

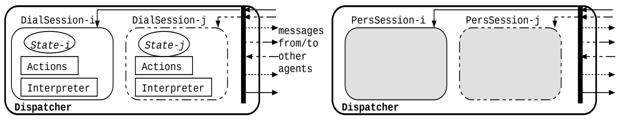

Figure 1: Parallel sessions within two SETA agents.

5 Management of parallel user sessions

The multi-user access to the Web store is managed by performing the session tracking within the Session Manager (a Servlet), and forwarding the session-specific messages to the agents of the architecture, so that they can process such messages and perform the related activities.

Each SETA agent is composed of an interface, the "Dispatcher", which handles the delivery and reception of messages, and of a set of *user-session agents* which maintain the session-dependent environments and carry on the activities to be performed within each user session; e.g., in Figure 1, the Dialog Manager and the Personalization Agent use two user-session agents each: "DialSession-i,j" and "PersSession-i,j". Each dispatcher acts as a wrapper and supports a uniform communication with the other SETA agents, by separating the communication flows related to the active sessions and forwarding incoming messages to the appropriate user-session agent. Moreover, the dispatcher creates and removes user-session agents, depending on the users connected to the store. The presence of dispatchers and user-session agents supports a simple management of session-dependent activities: user-session agents have separate

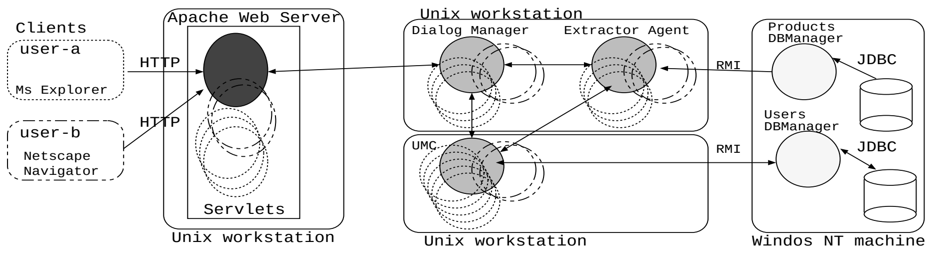

Figure 2: Parallel threads in the management of multiple user sessions.

internal states and work in parallel threads of execution within a SETA agent, performing services and activities related to the active sessions in an independent way.

6 Communication among SETA agents

As the number of distinct roles in the system architecture is fixed and well-determined, each agent knows which agents have to be contacted for requesting services and only needs their references; thus, there is no need to exploit middle agents: the Session Manager communicates such references after the creation of agents.

Our approach is integrated in the environment of the ObjectSpace Voyager tool [ObjectSpace, 2000], which we used to wrap the SETA agents, enabling them to run in parallel and to communicate via synchronous and asynchronous messages. The messages can be mapped to a subset of the KQML performatives. At the moment, the SETA agents do not need to communicate externally, therefore we did not comply with the FIPA ACL. According to the Voyager specification, the SETA agents exchange messages whose parameters contain, respectively, the name of the method to invoke and the array of its arguments. Voyager transforms the parameters of a message in a Java method call, so that the invocation of methods offered by a SETA agent is straightforward.[1]

Each Dispatcher is a Voyager Object and handles the incoming messages in parallel threads of execution, invoking the appropriate user-session agent to perform the requested task. As more than one message related to the same user session can be received at the same time, a user-session agent can perform parallel tasks; this fact raises mutual exclusion issues, resolved by inhibiting concurrent accesses to shared resources by means of Java synchronization facilities.

Figure 2 shows an example where two browsers ("user-a", represented as a dotted rounded square, and "user-b", a dashed one) access the system. The black oval represents the Servlet running within the HTTP server and

[1]For instance, an asynchronous message without return value, such as the following one: OneWay.invoke(agtReceiver, methodName, methodArgs); is translated by Voyager into: agtReceiver.methodName(methodArgs[0],methodArgs[1]).

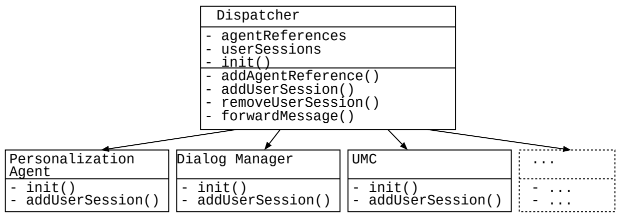

Figure 3: Class hierarchy defining a SETA agent.

the grey circles represent SETA agents: we have shown only the Dialog Manager, Extractor Agent and UMC. The figure shows multiple threads, related with the two active user sessions ("user-a", "user-b"), running within the HTTP Server and the SETA agents; the lines used to depict the threads correspond to the related sessions.

7 Design of a SETA agent

Figure 3 shows the class hierarchy defining the agents: the "Dispatcher" class specifies that a SETA agent has a list of references to the SETA agents it may send messages to ("agentReferences"); moreover, it has a set of user-session agents ("userSessions"). The class also offers the methods for initializing a SETA agent ("init()"), setting references of other SETA agents ("addAgentReference()"), creating and removing a user-session agent ("add/removeUserSession()") and forwarding messages to a user-session agent, to process a session-dependent request ("forwardMessage()"). The Dispatcher uses the communication facilities offered by Voyager for interacting with the other SETA agents.

Each SETA agent extends the "Dispatcher" class and may override the definition of its variables and methods. This is very useful for the definition of the user-session agents: to support the development of heterogeneous agents, specialized in the execution of different tasks, the SETA agents have to exploit user-session agents based on different technologies and designed following different approaches. For instance:

- In addition to the provision of services to the other SETA agents, the activities carried on by the Dialog Manager and the UMC include, respectively, the conditional execution of state transitions representing the evolution of the interaction with the user and the autonomous management of internal tasks. To satisfy these requirements, we have designed action-based user-session agents, where the agent state and the tasks to be performed are explicitly represented. Tasks are described as actions with preconditions determining the state conditions where they can be performed. In the Dialog Manager, the declarative representation of tasks supports an easy definition of the admissible state transitions in the interaction with the user. In the UMC, the presence of an interpreter selecting and performing actions, given the agent state, enables the agent to autonomously trigger its own internal activities, as soon as they can be performed [Ardissono *et al.*, 2000].

- In contrast, the action-based formalism is not suitable to other SETA agents, such as the Personalization Agent. In fact, the offered services can be handled by user-session agents responding to method invocations associated to potentially complex, but deterministic, tasks. Moreover, as such services must be based on the most recent information about the interaction with the user (situation of the user model, etc.), these agents do not need to explicitly manage an internal state: in fact, they have to retrieve such information from the other SETA agents, at each incoming request.

Figure 4 shows the class hierarchy of the action-based user-session agents defined in SETA. The "ActUserSessionAgent" class specifies their variables and generic behavior: it defines an explicit "state", representing the session-dependent data, an action list ("actionList") specifying the list of actions they can perform, and a pending list ("pendingList") used to store the suspended tasks. Moreover, the class offers an initialization method ("init()") and the "interpreter()". The user-session agents extend this class by redefining their own state (which may contain specific session-dependent data) and the action list, where their actions are listed. For clarity, the figure also shows the "Action" class, which provides the generic definition of an action specifying the basic components, i.e., the action type, preconditions, body and the method to check the preconditions, when the action is selected for execution; see [Ardissono *et al.*, 2000] for details.

8 Integration of heterogeneous software

We have integrated in the SETA agents external software tools for the management of very specific tasks. The modularity of the architecture enabled us to make them harmonically cooperate with the other components of the SETA agents. For instance, the UMC exploits the JESS rule-based system [Sandia, b] to maintain a structured

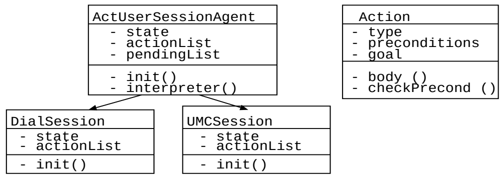

Figure 4: Class hierarchy defining an action-based user-session agent.

representation of the interaction context and to produce a synthetic description of the user's behavior: such description is then used within a Bayesian Network to reason about the user's interests and preferences.

The Natural Language Generator module of the Personalization Agent exploits JTg2 [DFKI and CELI, 2000] to generate the linguistic descriptions to be included in the Web pages presenting products and items. The JTg2 engine performs efficient and flexible generation in natural language. In its simplest conception it takes an object as input and returns a string. The crucial source of information is represented by the grammar rules: to integrate the engine in a new application, rules mapping the input object to the string have to be defined. The rules are augmented "if-then" statements spanning the input object top-down, left to right; to perform their task, the rules can access external modules that can be plugged into JTg2 to increase its own flexibility. In SETA, we have designed and implemented a set of NL grammar rules and two external modules: a test performer, which enables the engine to perform complex tests on the input object or on external sources, and a parameter getter, which enables the insertion of strings based on data coming from external sources. The input is a complex Java object representing a product and the output is the NL description to be included in the Web page. The grammar rules are used to fill in the slots in the templates defining descriptions.
JTg2 offers applicability tests to specify when a rule can be applied. In SETA, such tests are handled by the test performer and enable the grammar rules to be partially context-sensitive. In fact, they are essential to provide the NL Generator with the personalized content of the descriptions. Such content is represented by a selection of features, provided as a sorted list by the Personalization Agent on the basis of the contents of the user model. The applicability tests are used in SETA to integrate content personalization and NLG. In fact they make the NL Generator sensitive to the variation of input data (represented by the personalized list of product features to be described) and directly to the data in the user model; the NL Generator module, generates different linguistic descriptions, depending on the user's domain expertise. The use of NLG techniques reduces the amount of precompiled information to be defined at configuration time,

as the information about products is stored in a unique internal format. Moreover, it supports the generation of multilingual text (Italian and English) and the production of descriptions tailored to different user characteristics.

9 Technical details

The SETA system is implemented in Java (JDK 1.2) and is based on a Three-Tier architecture. The first tier runs on standard Java-enabled browsers, such as Netscape Navigator and Microsoft Explorer. In the second tier, the communication with the Web is supported by the Apache HTTP Server; moreover, we have exploited the functionalities offered by the Servlets to track the interaction with browsers. The bulk of the system resides in this tier and runs on a Unix environment: the SETA agents are distributed on two workstations for experimental purposes. The third tier includes the databases storing data about products and users and resides on a Windows NT computer. As shown in Figure 2, the communication between the second and the third tier is based on RMI (for historical reasons), while the agents in the middle tier communicate by exchanging Voyager messages.

10 Comments

10.1 Other three-tier architectures

As *n-tier* architectures are a popular solution for complex Web-based systems, the relation between our architecture and the other approaches has to be discussed, with specific reference to frameworks like J2EE [Sandia, a], which are industry standards for developing Web-based systems. The most important differences are in the organization of the middle tier of such architectures:

- The middle tier of SETA exploits Servlets for the communication with the Web, while it relies on the facilities offered by Voyager to exploit permanent distributed objects (SETA agents), which cooperate to carry on the interaction with the user. The whole logic for the generation of the Web pages resides in such permanent objects, with a specific object, the Dialog Manager, in charge of the selection of each interaction step, i.e., of choosing the next page to be produced, on the basis of the whole interaction context and of the last action performed by the user.
- The middle-tier of the J2EE framework uses Java Server Pages (JSPs) for the communication with the Web, and it uses Enterprise Java Beans (EJBs) to implement the other modules of the system.

The combined JSP-EJB paradigm is suited to a page-centric Web based system, where the interaction with the user is mostly based on the presentation of forms to be filled in and of pages containing the results of a (possibly complex) query. The typical content of such pages is the presentation of the results of a query to a database.

In contrast, the interaction with a Web store developed using SETA is not limited to a question-answer sequence: the user can navigate the catalog handling parallel contexts, searching for goods addressed to several beneficiaries during the same session. The system maintains parallel navigation contexts and supports the user in the switch among them. Moreover, the system can take the initiative and suggest alternative products to be analyzed, possibly interrupting the user's navigation. All these functionalities require the presence of an agent specialized in the management of the dialog with the user: this agent has to identify the next interaction step by using a declarative representation of the admissible turn sequences (e.g., a Finite State Automaton).

Of course, within a commercial application development, JSPs have a noticeable advantage with respect to pure Servlet-based approaches: the exploitation of JSPs facilitates the cooperation between Web authors and Java developers, letting Web authors concentrate on the development of HTML templates within the JSPs and the Java developers write the EJBs implementing DB access, legacy system interfaces, and so forth.

10.2 Technical user-interface issues

Although the exploitation of Applets has enhanced the functionalities offered by the user interface of SETA, we have experienced serious drawbacks. The most relevant problem is the fact that, to run an Applet containing non basic User Interface components (SWING), the browsers need to use plug-ins, sometimes incompatible with those installed on the computers. This requirement could make the access to the Web store complex and time-consuming, seriously reducing the portability of the system.

A second drawback concerns the combined use of Servlets and Applets within a system. On the one hand, the Servlets used to send HTML code to browsers can only receive String return values; on the other hand, for security purposes, Applets downloaded outside a LAN can only return values to the same HTTP Server from which they are downloaded, and firewalls forbid the communication with RMI servers. Together, these constraints impose that, whenever complex objects have to be returned by an Applet as the results of the decentralized interaction with the user, different Servlets within the same HTTP Server have to be exploited in the same application: one Servlet will forward Web pages to the browser and the other one will catch the complex return values produced by Applets. This approach has a subtle impact on the capability to track the state of the user sessions because it introduces parallel interaction flows between the browsers and the HTTP Server.

11 Conclusions

We have provided architectural and technological insights of SETA, a prototype toolkit for the development of adaptive Web stores developed at the CS Department of the University of Torino. While this architecture has been described at the abstract level in [Ardissono *et al.*, 1999; 2000], this paper specifies the implementation of the system and the class hierarchy underlying the definition of the agents composing the multi-agent architecture.

In the development of our system, we exploited a basic and light agent-building tool, such as Voyager, to manage a seamless communication among agents, but we preferred to design our own infrastructure for developing the system agents because, as SETA is a specialized architecture for the creation of Web stores, it does not need the full capabilities offered by general-purpose agent-building tools, which typically provide facilities for agent communication, coordination, self-diagnosis, mobility, and many other functionalities. For example, SETA does not support the development of open systems interacting with middle agents. Thus, popular coordination models, such as ABS [Barbuceanu and Teigen, 1999], exceed the demands of our application example, because they are focused on a more complex issue, i.e., describing the external behavior of social agents. Other tools for the development of multi-agent systems, such as DECAF [Graham and Decker, 2000], seem to exceed our needs as well, as they support complex activities such as the coordination of a multiagent system to reach non-local goals and real time flexibility in the execution of tasks. In our system one agent is associated to each main role of the architecture; thus, we do not need to exploit schedulers for distributing tasks among alternative agents.

Scalability is a critical aspect and concerns several issues, among which load balancing. Being SETA a prototype, we did not explicitly address such aspect in our implementation; however, this problem can be bypassed by using the Voyager 3.3 Professional edition, which provides services that would take care of this issue.

We are now working to enhance the configurability of SETA, to support its instantiation on new domains for creating new Web stores, or generic recommender systems. A graphical tool currently enables the store designer to introduce the knowledge about customer classes (Stereotype KB) without writing any Java code. Moreover, the knowledge base containing information about products and their features, is automatically created by the system, given the structure of the Products DB (which contains a classification of items in product classes). Finally, we are integrating XML-based representations of the content of the Web pages generated by the system: different page types are defined in a DTD (Document Type Definition) and XSLT (eXtended Stylesheet Language Transformations) are used to produce the final user interface, on the basis of the personalized content of the page, encoded as an XML object: in the simplest case, such interface is generated as HTML code.

References

[Ardissono and Goy, 2000] L. Ardissono and A. Goy. Tailoring the interaction with users in web stores. *User Modeling and User-Adapted Interaction*, 10(4):251–303, 2000.

[Ardissono *et al.*, 1999] L. Ardissono, C. Barbero, A. Goy, and G. Petrone. An agent architecture for personalized web stores. In *Proc. 3rd Int. Conf. on Autonomous Agents*, pp. 182–189, Seattle, WA, 1999.

[Ardissono *et al.*, 2000] L. Ardissono, A. Goy, G. Petrone, and M. Segnan. Configurability within a multi-agent web store shell. In *Proc. 4th Int. Conf. on Autonomous Agents*, pp. 146–147, Barcelona, 2000.

[Barbuceanu and Teigen, 1999] M. Barbuceanu and R. Teigen. Higher level integration by multi-agent architectures. In P. Bernus, ed., *Handbook of Information System Architectures*. Springer Verlag, 1999.

[DFKI and CELI, 2000] DFKI and CELI. JTg2. www.celi.it/english/tecnologia/tecLing.html/, 2000.

[Giampapa *et al.*, 2000] J.A. Giampapa, M. Paolucci, and K. Sycara. Agent interoperation across multiagent system boundaries. In *Proc. of 4th Int. Conf. on Autonomous Agents*, pp. 179–186, Barcelona, 2000.

[Graham and Decker, 2000] J. Graham and K. Decker. Tools for developing and monitoring agents in distributed multi agent systems. In *Proc. of the Agents'2000 workshop on Infrastructure for scalable multi-agent systems*, Barcelona, 2000.

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

[Sandia, a] Sandia National Laboratories. Java 2 Platform Enterprise Edition. http://java.sun.com/j2ee/.

[Sandia, b] Sandia National Laboratories. JESS, the Java Expert System Shell. http://herzberg.ca.sandia.gov/jess/.

[Macho *et al.*, 2000] S. Macho, M. Torrens, and B. Faltings. A multi-agent recommender system for planning meetings. In *Proc. of the Agents'2000 workshop on Agent-based recommender systems (WARS'2000)*, Barcelona, 2000.

[ObjectSpace, 2000] ObjectSpace. Voyager. http://www.objectspace.com/index.asp, 2000.

[Riecken, 2000] D. Riecken, editor. *Special Issue on Personalization*, volume 43. 2000.