Structureless, Intention-guided Web Sites: Planning Based Adaptation

Matteo Baldoni, Cristina Baroglio, Viviana Patti

Dipartimento di Informatica – Università degli Studi di Torino
C.so Svizzera, 185 – I-10149 Torino (Italy)
Tel. +39 011 6706711, Fax. +39 011 751603,
E-mail: {baldoni,baroglio,patti}@di.unito.it
URL: http://www.di.unito.it/~alice

Abstract. The great variety of users of the services available on the internet raised the problem of finding flexible forms of presentation and interaction, which depend on the specific user's characteristics and needs. In this article we present an approach to adaptation, which exploits the reasoning capabilities of a rational agent. The key idea, which makes this approach orthogonal to personalization based on the "user model", is that the system adopts the user's intentions during the interaction. Then the system exploits its planning capabilities to dynamically generate a web site, adapted to the current user's needs. For the sake of understanding we also discuss a possible application of the approach to the construction of a virtual tutor.

1. Introduction

Many of the most advanced solutions on web site adaptation to the user, such as (Ardissono *and* Goy, 1999a; Marucci *and* Paternò, 2000; De Carolis, 1998), start from the assumption that adaptation should focus on the user's characteristics; in different ways, they all try to associate the user with a reference prototype, the "user model", to which the presentation is adapted. By doing so, such approaches catch important aspects connected to the personality and the general interests of the user but, in our opinion, they underestimate the role that the user's intentions and needs, which may change at every connection, have in order to achieve a real adaptation.
In this paper, we present an approach to web site adaptation which uses an agent logic programming language, called DyLOG (Baldoni *et al.*, 1998; Baldoni *et al.*, 2000a), for building cognitive agents that dynamically generate a site based on the user's goals and constraints. When a user connects to a site managed by one of our agents, (s)he does not access to a fixed graph of pages and links but (s)he interacts with a rational agent that, starting from a knowledge base specific to the site and from the requests of the user, builds an ad hoc presentation structure. We obtain run-time adaptation by exploiting the capability of a rational agent to reason about its actions and to make plans. So the problem to generate a web site adapted to user's needs is interpreted as a planning problem. In this sense, our web sites are *structureless*. Their structure corresponds to the plan built for pursuing the user's goal. Thus adaptation occurs at the navigation level rather than at the presentation level. Indeed, our focus is on the definition of the navigation possibilities available to the user and on determining which page to display based on the dynamics of the interaction.
This approach brings along various innovations w.r.t. what can be found in the literature. From a *human-machine interaction* perspective, the kind of adaptation that we propose is focused on the actual interests of the user for that specific connection and not, as it often happens in the user model approach, on his/her generic and cultural interests or past choices (past-oriented adaptation). From a web site design perspective, some advantage is expected for the *build-modify-update process*. In fact, while in order to modify a classical web site one has to change both the contents of the pages and the links among them, in our case the site does not exist as a structure, because the structure is built at the moment. All that we have is a data base containing the domain knowledge, whose maintenance is simple, and an agent program, that is not changed often.
Although our approach could recall some works on natural language cooperative dialogue systems, there are some differences. For instance, in (Bretier *and* Sadek, 1997) Sadek and Bretier proposed a logic of rational interaction for implementing the dialogue management components of a spoken dialogue system. This work is based, like DyLOG, on dynamic logic and reasoning capabilities on actions and intentions are exploited for planning dialogue acts. Our work, instead, is not focused on recognizing or inferring the user's intentions (which is also a very interesting task): the user's needs are taken as explicit input by the software agent, which uses them to generate the goals that will drive its behavior. The novelty of our approach is that we exploit

planning capabilities for building web site presentation plans guided by the user's goals rather than for dialogue act planning. The structure of the site is built by the system as a conditional plan, according to the initial user's needs and constraints. The execution of the plan by the system corresponds to the navigation of the site, where the user and the system cooperate for finding a solution to the user's problem.
We chose DyLOG as an agent programming language due to two of its main characteristics. The first is that, being based on a formal theory of actions, it can deal with reasoning about action effects in a dynamically changing environment and, as such, it *supports planning*. The second is that the logical characterization of the language is very close to the procedural one, and this allows to *reduce the gap* between theory and use.
In order to check the validity of the proposed approach, we have implemented a client-server agent system, named WLog. In this application the users connect to the structureless web site for asking for support in some task. As an example, we are currently implementing a virtual seller that helps clients to assemble a computer, which is good for their needs (Baldoni *et al*., 2000b; Baldoni *et al*. 2001), but we will show in the next section that the same approach can be exploited in other, apparently very different, applications. The idea is that the user interacts with a software agent for solving a problem. Similarly to what happens in an interaction between humans, the solution is obtained thanks to a dialogue, in this case guided by the agent (see next section for a an example). Requests, information, and proposals are presented to the user in the form of web pages.
Technical information about the system can be found at `http://www.di.unito.it/~alice`.

2. An Example Application: the Virtual Tutor

In a parallel work, we are using WLog to tackle an e-commerce application: to build a "virtual seller" that helps users to assemble computers depending on which use that computer will have. To this aim the agent system adopts the user's goal in order to plan a solution. Another application in which a similar interaction is useful is the construction of "studiorum itinera" that depend: (1) on the competence a student wants to acquire, (2) on his/her current competence, (3) on the material available in the system's store and (4) on student's time constraints. Consider the following example ("Tutor" is the agent system):

Declaration of the user's main intention

Student: I wish to know something about programming languages.

Tutor (asking): Are you interested in an *advanced* or in an *introductory* course?

Student: I'm a beginner. I would like to start with an introductory course.

Definition of the learning goals

Tutor (thinking): *Let me think, for an* introductory *course, first he needs a training module about the principles of programming, then some module of exercises to test the newly acquired knowledge. At this point he can continue by studying Object Oriented programming; afterwards, he will have to use an OO exercise module, and at last he will be ready to study some specific language. Let's list what he should learn…*

Acquisition of the user's preferences

Tutor (asking): This is a sketch of the knowledge that you should acquire in order to reach your learning target, is there something that you already know?

Student: I already had a crash course in programming, so I would skip the part about the basic principles; I would start from OO theory.

Tutor(asking): Would you prefer a short course (maximum 20 hours), a medium course (maximum 40 hours) or time is not a problem?

Student: I'd rather choose a medium length course, I work part-time.

Planning

Tutor (thinking): *I need to* ***plan*** *according to his actual competence, time constraints, and the availability of training modules…*

Execution

Tutor (explaining): Well, I have many solutions that could satisfy you. For learning OO programming theory, you can choose between a 20 hours module by prof. Smith or a 10 hours module by prof. Black. Which one do you prefer?

Student: I prefer the 20 hours course by prof. Smith.

Tutor: Afterwards you can practice with the exercises contained in a 5 hours module by prof. Green; you could end your course by studying one of the following OO languages *[list]*. Which one do you prefer?

Student: I heard that my colleagues use java; I choose the java 10 hours course by prof. Giovannetti…

The student expresses his/her desire. The tutor starts a dialogue aimed at fixing both a set of subgoals and a set of constraints (such as the overall time the student can devote to the course); afterwards, it builds a conditional plan that will allow the student to grow knowledge about programming languages; the execution of the plan leads to show the web pages which correspond to the different actions in the conditional plan (see Sec. 4). The process is similar to the one presented in (Baldoni *et al*. 2001).
An agent system of this kind could be a valid interface for an on-line library of training modules (in the line of commercial systems, such as Competus Framework). Modules can be tutorials, CBT tools, links to web pages, etc. and have the most various origin. Each module has an associated description about the prerequisites necessary to use it and what it allows to learn. The agent system works on the meta-knowledge in order to build plans. Such a system would be extremely useful for all those persons who cannot attend regular classes: they could download the suggested modules and use them when and where they can.
Some examples of persons who could benefit of a similar system are students affected by serious illnesses (e.g. cancer-affected students often have to spend long periods of time at home or in a hospital), persons who want to increase their education but have a full-time job, persons in jail. Indeed, a system of this kind, with a reach enough module set, could serve different categories of people. Plans could therefore depend also on the category the user belongs to. In this way we would have a richer form of adaptation: on one hand, we would have a personalization according to the category (user model), while on the other we would have a more detailed adaptation to the specific user and situation (the study plan).

3. Modeling the Virtual Tutor as a Rational Agent in DyLOG

The language that we use for specifying our *virtual tutor* is DyLOG. It is based on a logical theory for reasoning about actions and change in a modal logic programming setting. It allows one to specify a rational agent's behavior by defining both a set of *simple actions* that the agent can perform, some of which are *sensing* and *suggesting* actions that allow to interact with the user, and a set of prolog-like *procedures* which build complex behaviors upon simple actions. Simple actions are defined in terms of preconditions and effects on a *state*, which consists in a set of fluent formulas representing the agent beliefs. Indirect effects of actions are expressed by means of *causal rules*. See (Baldoni *et. al*, 2000a, 2001) for details on the language and on its use for agent programming. The behaviour of our *virtual tutor* is captured by a collection of procedures and it is driven by a set of goals. The top level procedure, *advise* is defined as follow:

advise **is** *ask_student_competence*; *ask_available_time*;
plan(*combine_courses*, *total_time*(H) $\wedge$ *student_time*(T) $\wedge$ ($H \leq T$),*Plan*); *Plan*.

The virtual tutor starts by interacting with the student in order to find (and adopt) his/her learning goals, by asking if (s)he is interested in an *advanced* or in an *introductory* course and by checking if the user already has some of the competences which are the target of a standard program (*ask_student_competence*). Then it gets information about student's time constraints (*ask_available_time*). By using this information, the knowledge on the available training material and its expert competence about how to combine the courses, it starts to plan how to reach the goals of the program, predicting also future interactions with the student. Planning is needed to find *study itinera* by taking into account two interacting goals: the goal of combining courses into a program which satisfy the student learning needs and the goal to consider only programs affordable by the student's time contraints. Finally, the agent executes the conditional plan P resulted from the planning process.
The way the agent combines the courses into a program is specified by procedure *combine_courses* that, until the program is believed complete, tries to achieve the goal of getting a still missing training module.

combine_courses **is** *program_complete*?.
combine_courses **is** ¬*program_complete*?; *achieve_goal*; *combine_courses.*

Note that only when all of the goals to get the necessary training modules are fulfilled combining available courses, the main goal to have a program to propose to the student is reached and the program is considered complete. Until there is still a goal to fulfill, the program is considered not complete (this is expressed by the causal rule: ¬*program_complete* **if** *goal(X)*). Indeed, we assume the behaviour of a rational agent to be *driven by a set of goals*, which are represented as fluents having form *goal(F)*. The system detects the goals based on student's inputs and its expert competence about learning and courses combination. Initially the tutor does not have explicit goals, because no interaction with the student has been performed. The student's inputs are obtained after the first interaction phase (see *advise*) and they generate a set of goals that the agent has to achieve to compose a suitable program. In the language, we model this by describing the adoption of a goal as the indirect effect of requesting user's preferences. For instance, the suggesting action *ask_program_level* (which is a simple action of *ask_student_preferences*) asks if the student is interested in an advanced or in an introductory course. This action has as indirect effect the generation of the goal to have a program leading the student to reach a given level of competence:

ask_program_level **suggests** *requested*(*competence*(*Level*)).
goal(*has*(*competence*(*Level*))) **if** *requested*(*competence*(*Level*)).

Let us suppose the student is interested in an introductory course (fluent *requested (competence(introductory))* is known by the agent), then, the causal rule above will generate the goal *goal*(*has*(*competence*(*introductory*))). By means of an appropriate instantiation of the following causal rule, this main goal will generate a set of sub-goals for selecting the training modules that could allow the student to acquire the desired competence:

goal(*user_knows*(*Subject*)) **if** *goal*(*has*(*competence*(*Level*))) $\wedge$ *part_of*(*competence*(*Level*),*Subject*).

Roughly, this is the way in which the agent initializes the set of goals[1] that it will try to solve by means of planning, which is done as described in (Baldoni *et al.*, 2001). The extracted plan is one possible execution of procedure *combine_courses*. Indeed we take advantage of a specific instance of the *planning problem,* by looking for a possible execution of procedure *p* leading to a state where some condition *Fs* holds. In this case the search space for an action sequence leading to the goal is naturally constrained by the procedure definition. Fig. 1 reports a conditional plan that could be obtained when the agent plan for composing an introductory program with medium length, i.e. maximum 40 hours. Each box represents a simple action, which will cause the creation of a web page. Those corresponding to sensing and suggesting actions allow an interaction with the user, and can be seen as queries for inputs whose values cannot be known at planning time. Each branch of the plan corresponds to one of the possible input values.

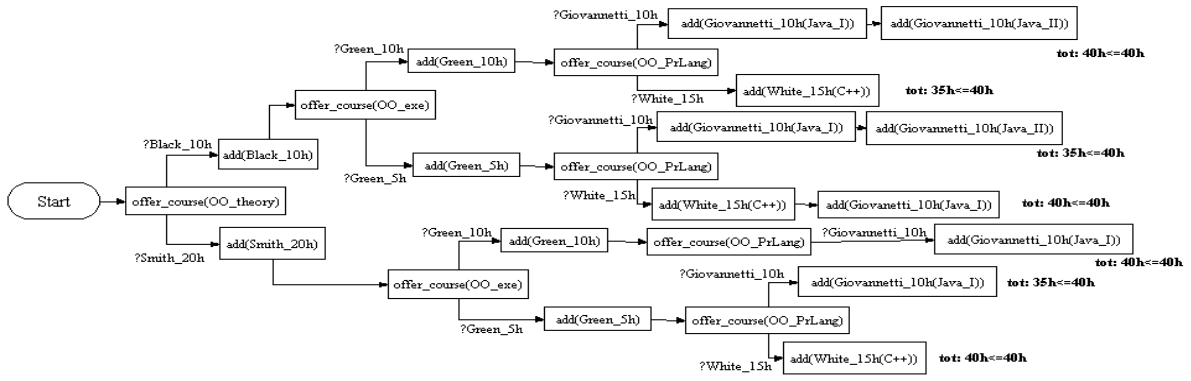

Figure 1. Result of *plan(combine_courses, total_time(H)* $\wedge$ *student_time(40)* $\wedge$ *H* $\leq$ *40, Plan).*

4. WLOG: System Architecture

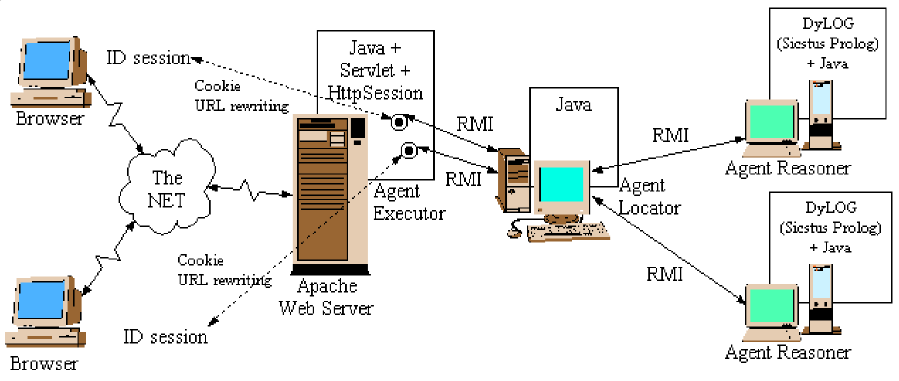

Figure 2. WLog multi-agent architecture

WLog is the system that we developed in order to verify the feasability of the intention-driven approach. The novelty respect to previous implementations is that it has a *multi-agent architecture*, which is sketched in Fig.

[1] Observe that agents use a *blind commitment strategy* to their goals

2. Briefly, the system consists of two kinds of agent: *reasoners* and *executors*. A DyLOG reasoner generates conditional presentation plans; once built the plan is executed. Actual execution it is the task of the executors, which are Java servlets embedded in an Apache web server. *Executing* a conditional plan implies following one of the paths; only the part of the site corresponding to this path will actually be built. The execution of the actions in the path consists in showing one or more web pages to the user; in particular, when a page that corresponds to a branching point is shown, a feedback from the user is required. Therefore, the execution of each action in the plan consists of the following steps: 1. the reasoner sends a message to the executor; 2. the executor produces a proper HTML page; 3. the page is shown to the user and a feedback is asked; 4. the executor sends the feedback to the reasoner; 5. the reasoner adds the new fact to the knowledge base and takes it into account for passing to the next step: sending to the executor the information about which page to show next. All unselected alternatives are forgotten. The connection between reasoners and executors has the form of message exchange in a distributed system; message exchange is FIPA-like (FIPA, 1997). Each agent is identified by its "location" which can be obtained by other agents from a facilitator ("Agent Locator" in Fig. 2). The interaction between the user and WLog starts with the declaration of the user's goal. The executor works as an interface between the reasoner and the user. It looks for a free reasoner; if one is available, from that moment till the end of the connection will be dedicated to serve that specific user.

5. Conclusion

In this paper, we have shown how to implement web applications, where a real-time adaptation is required, by exploiting planning capabilities of an agent logic language. Adaptation is obtained by dynamically generating presentations guided by the interaction with the user. The observation that user's goals may change at every connection, which motivates our proposal, seems to be particularly true for web-sites implementing *recommendation systems*. As representative of this category, we chose the virtual tutor case study. Of course, the planning task that we described is just one of the tasks that a real tutoring system should have. For instance, a real system should also be able to check the progress of each student and, if necessary, perform some replanning; all these problems are, however, out of the scope of the current example.

References

Ardissono, L. and Goy, A. (1999). Tailoring the interaction with users in electronic shops. In *Proc. of the 7th International Conference on User Modeling*, pages 35-44, Banff, Canada. Springer-Verlag.

Baldoni, M., Giordano, L., Martelli, A., and Patti, V. (1998). A Modal Programming Language for Representing Complex Actions. In the *Post-Conference Workshop DYNAMICS'98*, pages 1-15.

Baldoni, M., Giordano, L., Martelli, A., and Patti, V. (2000a). Modeling agents in a logic action language. In *Proc. of the Workshop on Rational Agents, FAPR'00,* London, September 2000. To appear.

Baldoni, M., Baroglio, C., Chiarotto, A., Martelli, A., and Patti, V. (2000b). Intention-guided Web Sites: A New Perspective on Adaptation. In Proc. of the *6th ERCIM Workshop UI4All*, pages *68-82*, Florence, Italy.

Baldoni, M., Baroglio, C., Chiarotto, A., and V. Patti (2001). Programming Goal-driven Web Sites using an Agent Logic Language. In *Proc. of the Third International Symposium on Practical Aspects of Declarative Languages*, volume 1990 of LNCS, pages 60-75, Las Vegas, Nevada, USA. Springer-Verlag.

Bretier, P. and Sadek, D. (1997). A rational agent as the kernel of a cooperative spoken dialogue system: implementing a logical theory of interaction. In *Proc. of ATAL-96*, LNAI 1193, pp. 189-204. Springer-Verlag.

FIPA (1997). Agent Communication Language. FIPA 97 Specification, Foundation for Intelligent Physical Agents, `http://www.fipa.org`.

De Carolis, B.N. (1998). Introducing reactivity in adaptive hypertext generation. In H. Prade, editor, *Proc. of the 13th European Conference on Artificial Intelligence, ECAI'98*, Brighton, UK. John Wiley & Sons.

Marucci, L. and Paternò, F. (2000). Designing an Adaptive Virtual Guide for Web Application. In Proc. of the *6th ERCIM Workshop UI4All*, pages 57-67, Florence, Italy.

McTear, M. (1993). User modelling for adaptive computer systems: a survey on recent developments. In *Journal of Artificial Intelligence Review*, 7:157-184.