Offering Existing AI Planners as Web Services

Ronny Hartanto
B-IT Bonn-Aachen International Center for Information Technology
D-53754 St. Augustin
ronny.hartanto@fh-bonn-rhein-sieg.de

Joachim Hertzberg
Institut of Computer Science
University of Osnabrück
D-49069 Osnabrück
hertzberg@informatik.uni-osnabrueck.de

***Abstract*— Robots or other agents with some need for action planning functionality may not insist on running their planners on-board. Web services technology offers a very flexible way of connecting planning clients to planning servers, disregarding the language in which the concrete planning system is implemented and its operating system requirements. A planning web service can then be used locally within an intranet or globally over the Internet. The paper describes our implementation.**

I. Motivation

Planning is a ubiquitous activity. It comes in many different forms, differing, for example, in the level of abstraction and in the future time window for which it makes sense to plan, given the limits of domain knowledge and of previously unknown domain dynamics. Yet, it makes sense for many agents to deliberate about their own future course of action or about actions to be done in coordination with others. Concrete behavior in time may then have more sources than just plans designed by past explicit deliberation, such as reflex-type behavior, opportunistic action improvisation, or canned (in the case of artificial agents: programmed) ways of doing things never put into question. However, we take it that deliberated plans are an important information source for acting in many agents.

The field of Automated Planning [1] has made quite some progress recently. Flagship systems and applications have been around for decades, but as a relatively recent development, planning technology has become widely available, and is has been broadening up from the concentration on classical situation-based, partial-order, complete-information, instantaneous-action view to a richer variety of domain assumptions under which automatically generated plans work. In particular, planning and scheduling are finally being blended, leading to practical systems for a wealth of applications.

So, planning makes sense for agents, Automated Planning is a well-advanced field – then let them plan! In fact, that is exactly what we intend to do for the case of autonomous mobile robots as agents. However, a number of issues need to be solved before mobile robots will smoothly execute plans generated by your favorite AI planner and/or scheduler. These issues are being addressed under the topic of plan-based robot control [2]. They include the need to find plan formats suitable for both automatic plan generation and monitored autonomous execution, find ways of blending these plans with reactions needed to respond to asynchronous events in short (e.g., 10 ms) control cycle times, to update symbolic world models based on sensor data, and more.

This paper deals with our practical solution to a particular problem that we have in plan-based robot control, but applies to more artificial agents than robots only. We want our robots to use existing planners as they are, with no regard of their programming languages and operating environments – the choice of planner should be guided by its functionality rather than implementation or operating system details. The planner should run asynchronously off-board the robot, saving precious on-board

processor capacity. The idea is that the robot sends its problem description in a format suitable for the called planner, and gets back a plan later, using the remote planning time for doing whatever it has on its agenda.

Our solution consists of encapsulating planners as Web Services. This is a rapidly evolving technology, based on the XML standard and regular Internet technology. The web service provides a solution for working on different operating systems or programming languages. It uses XML for mapping objects into text-based data that can be easily transported over normal web servers by using HTTP.

A planner wrapped as a web service will run in its original environment and language, so no porting effort is required. The web service will get the request over a normal web protocol, deliver it to the planner and return the result to the client, which is a robot in our case, but may be any artificial agent in general. Thus, the client can be anywhere in the Internet or intranet, which is connected to the server, on any operating system. The client may run any programming language as far as it supports XML technology. Of course, it needs to be connected with the net, which is the case for our robots, who are accessible by WLAN.

The rest of the paper describes the architecture of the encapsulation and the services provided in more detail and sketches our future ideas with it.

II. Architecture

A web service, according to w3.org, is "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." [3]

What we do here is encapsulating an AI planner or planners in a web service. The client request consisting of a planning domain and problem description will be sent in XML format to a web service. The web service parses these data and calls the planner with the respective parameters in its own, possibly proprietary syntax. The results from the planner will be sent back to the client in an XML format. Apache-Tomcat is used in this work for serving the HTTP requests. The SOAP service is provided by the Apache-Axis, which is integrated in the Apache-Tomcat server.

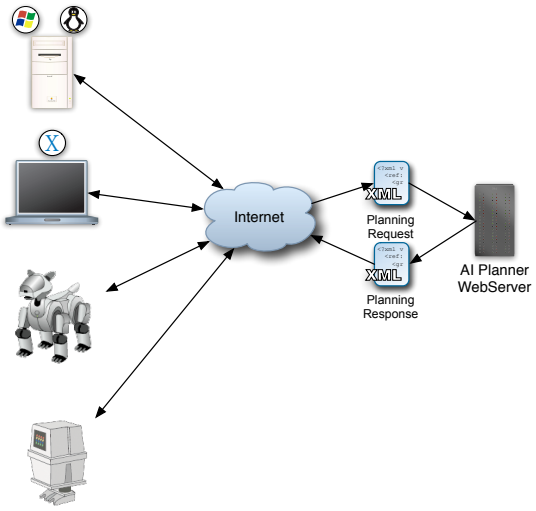

Fig. 1
AI Planner WebService with several clients

Figure 1 sketches this scenario for an AI planner web service with four clients. The clients can be embedded agents or normal notebook or desktop computers. The clients communicate with the server by sending and receiving XML documents through HTTP over Internet. The sketch includes just one server. In general, there should of course be many of them – typically including a planner server run at the planner developer's site.

Figure 2 gives a close-up of the AI planner web service as we have designed it. The web service is programmed in Java, thus can be run on different platforms. It essentially consists of several sub-services for handling client requests: AI Planner Manager, Session Manager, Configuration Manager, and Admin Service.

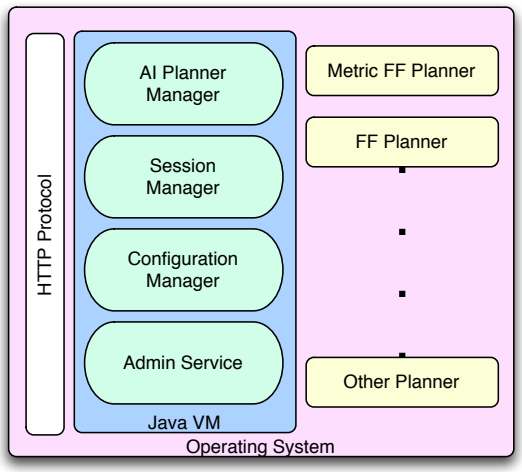

Fig. 2
AI PLANNER WEBSERVICE ARCHITECTURE. SEE TEXT FOR DETAILS ON ITS COMPONENTS.

We will now describe them in turn.

A. Planner Manager

The planner manager is the main service. It provides all functions related to the planning tasks, such as upload a domain, query planner list, and solve a problem.

As in a stand-alone planner, domain descriptions can be re-used for different problem descriptions. Therefore, the client can upload its domain to the AI planner web service, and store it there. In case that the domain is big, it can be divided and submitted to the server in parts. After getting the last part, the server will combine everything into a single domain file. With this feature, the client can have a collection of domains on the server.

There are two query tasks in this web service: `getPlannerList` and `getDomainList`. The `getPlannerList` function returns the available installed planners on the server. `getDomainList` returns the uploaded domains from the current client, who calls this function.

Two possible solve functions, `instantSolve` and `solve`, are available in this service. `instantSolve` provides a direct solution to the given domain and problem files in the parameters. It can be called from each client, without requiring the client to login to the system. It assumes that the problem to be solved is simple and can quickly or instantly be solved by the planner. The `solve` function can only be accessed if the client logs in to the system. The client calls this function with problem, domain name, and planner name as parameters. The domain name is one of the stored domains in the server. The planner name is in the available planners on the server. It is possible that the client selects more than one planner to solve its problem. The server will give the client a unique transaction ID generated based on the given parameters values.

Using the transaction ID, the client can get the solution from the server, once the planner has terminated. The function `checkSolution` serves for asking whether the planner is already done or not.

B. Session Manager

The web service uses an asynchronous communication between client and server [4]. Thus, a session manager is required for handling and tracking the client activities.

Each client can retrieve its own preferences, its own domain files and finished solutions from the server with the help of its session ID. The session ID is unique, created based on the client's username, password and login time. Before a client can use this web service, it must login to the server first to get its session ID. With this ID the server can keep track of each client. The clients can retrieve their uploaded domains and solutions from the previous session. The session ID is only valid for certain time. If a client has shown no activity for longer than a maximum time-out time, its ID will be expired. If it is expired, the client must login again to get a new session ID. There is a function to check the status of a session ID. The session ID is also expired if the client logs out.

The session manager has two administration tasks; register new users to the system and

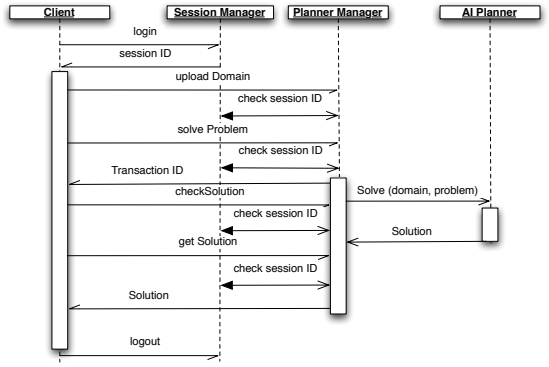

Fig. 3
PLANNER TASK SEQUENCE DIAGRAM

set session time-out. These tasks are available only for the administrator.

C. Configuration Manager

The configuration manager manages the server settings, such as planner commands and client's directories. The planner command is used internally within the web service, it returns the value or command related to a specific planner. It also manages how the planner gets the domain and problem on the server; manages the directories where the client data are stored, where the domain files and output files are generated; and finally manages the temporary directory for the planner.

D. Admin Service

The administrator can do additional tasks with the admin service. `getSystemInformation` returns information about the server and its properties. A user list can be fetched by using `getUserList`. The active clients or session can be queried using `getSessionList`.

E. Sequence Diagram

Figure 3 shows in a sequence diagram how the AI planner web service serves a client request. The client first has to login with its username and password. After successful login, the client will get an active session ID from the session manager. If the client wants to solve a problem of which the domain is not yet in the server, it must first upload the domain to the planner manager.

When the required domain is already on the server, the client can directly call the `solve` function on the planner manager with the domain name, the planner name and the problem as parameters. The planner manager will generate a transaction ID for this activity and return it to the client. The planner manager then calls the selected planner with the corresponding parameters. The output or solution from the planner will be stored on a file to be retrieved later.

The client can call `getSolution` with its transaction ID as parameter for retrieving the resulting plan from the planner manager. The client could also first call `checkSolution` to make sure that the planner has finished computing the solution. Finally, the client would logout from the system.

The client includes its session ID on each call, like `uploadDomain`, `solve`, `checkSolution` and `getSolution`. The planner manager consults the session manager for checking the client's session ID validity.

III. STATE OF THE IMPLEMENTATION

The AI planner web service has been built in Java, therefore it is operable on most operating systems. It has been used on Windows, Linux, and Mac OS X systems. The web service is installed on the department of Computer-Science (Knowlegde-Based System) server in the University of Osnabrück. Currently the server is behind the firewall, thus it is only accessible from the University intranet.

Currently, PDDL [5] is the domain description language that is supported. For testing the system, we have locally installed two planning systems available on the Web, namely, FF [6] and Metric FF [7].

The web service server runs on the Apache-Tomcat server with Axis. Currently the service has been tested with the Java-client only.

IV. Outlook

Our current implementation could be extended in a straightforward way in several directions. Here we mention two of them.

A. Beyond PDDL

Currently our planner web service implementation only supports PDDL-based planners. This is of course not necessarily so – what is required is just some fixed form of domain description, for which an XML syntax and a parser from XML to the concrete planner input syntax would have to be provided.

In the future, additional non-PDDL-based planners or planners conforming to newly emerging domain description standards could also be supported and integrated into this service, enhancing not only the zoo of different planning systems for a common domain description standard, but the types of planning that it makes available.

B. Service Brokerage and Load Balancing

As explained in Sec. II, the planners still could run on their original platforms without no porting or modification required. In fact, we do not envision huge planning system servers providing any planner that is out there as a central service. Rather, every planner web server should be lean, typically providing stable versions of the systems developed on site, and maybe a few others that are similar in operation requirements.

Would a planning client then have to know about all planning web servers that it might want to use? Of course not! The obvious solution is to make planning web services known to other planning web servers, so that every single server could also work as a broker for requests that it could not handle because it either does not run a requested planner or because it is currently overloaded. So rather than building huge software repositories, the situation would look as sketched in Fig. 4. A client could address any of the four servers for requesting planning services, and the request would be delegated to a server running the requested planner and with spare capacity.

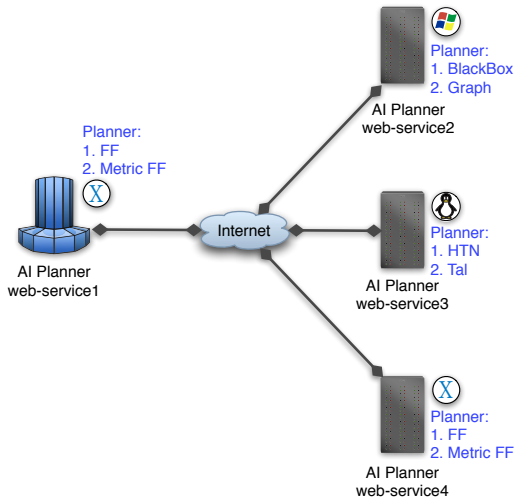

Fig. 4

Four AI planner web services that are able to function as each other's brokers and perform load balancing (servers 1 and 4).

References

[1] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Morgan Kaufmann Publishers, May 2004.

[2] M. Beetz, J. Hertzberg, M. Ghallab, and M.E. Pollack, editors. *Advances in Plan-Based Control of Robotic Agents*, volume 2466 of *LNAI*. Springer, Berlin, 2002.

[3] W3C Working Group. Web services architecture, February 2004.

[4] W3C Recommendation. SOAP version 1.2 part 1: Messaging framework, June 2003.

[5] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale University, Yale Center for Computational Vision and Control, October 1998.

[6] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *J. AI Research*, 14:253–302, 2001.

[7] Jörg Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.