Task-sensitive Adaptability for Software Agents

SANTTU TOIVONEN
VTT Information Technology
P.O.Box 1203, FIN-02044 VTT
FINLAND

HEIKKI HELIN
Sonera Corporation, Sonera R&D
P.O.Box 970, FIN-00051 Sonera
FINLAND

Abstract: – We present a model of an adaptive software agent. An agent implementing the model is capable of adjusting itself to different environments that may have features new to the agent. We explicate some of the properties we feel are useful for adaptability. The basic adaptability features could be abstracted to virtually any kind of agent—artificial or living one—that acts in heterogeneous environments. As an example, we consider a software agent functioning in a web environment, especially in the Semantic Web. The underlying idea is to defend adaptability as an important property for software agents. We concentrate particularly on the adaptation to domain-specific tasks.

Key-Words: – Agent, ontology, adaptability, task, communication

1 Introduction

The concept of agent can be analyzed and decomposed into properties that constitute it. There are number of definitions containing varying combinations of properties important for agency. Autonomy, reactivity, proactivity, social skills, mobility, adaptability and intelligence are among the most frequent properties in the literature (see e.g., [6]). With respect to this paper, adaptability is the most important property of these. Furthermore, we consider agent mobility and show that adaptability could be a relevant property of a mobile agent.

Adaptability stretches from simple adjustments of behavior with predefined parameters all the way to complex and unpredictable learning. An entity that can adapt to its surroundings represents more agency than its counterpart without the adaptation capabilities. An obvious issue is how fast the agent adapts its behavior to the new environment. In [9], the authors come to the conclusion, that if the environment change rate is low, agents that do more reasoning perform better than those that employ a more straight-forward behavior. On the other hand, if the world change rate is high, a straightforward behavior outperforms those agents that "waste" too much time on thinking. The bottom line is that the more environments differ from each other, the better adaptation capabilities are required from agents visiting them. There are number of different means for categorizing environments. For example, the environment can be static or dynamic, episodic or nonepisodic, and accessible or inaccessible [10].

Adaptability can be sensitive to facts or tasks. With fact-sensitive adaptability, we mean that the agent can learn new concepts and use them in its tasks. For example, an agent searching for publications in the web might have concepts "book", "article", and "magazine" in its ontology about publications. It might acquire new concept called "position paper" and include it in its ontology. However, in this paper we concentrate more on task-sensitive adaptability. This means, for example, that the agent searching for publications might learn some new means of performing searches. We feel that dynamic retrieval of domain-specific tasks, for example interaction protocols, is extremely important for effective adaptability. The tasks software agents perform can be considered as services. Agents in web environment thereby provide web services. There has been plenty of work around web services and semantic web in the past two years (see e.g., [14, 7, 1, 2]).

We consider an agent implementing our general model of adaptability acting in Semantic Web environment. The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation [1]. The crucial difference with the web of today is that the information is provided with semantics; the concepts appearing in Semantic Web are defined and given meanings.

The rest of this paper is organized as follows. In Section 2, we introduce concepts useful for modeling adaptive agents. These concepts are taken from human memory research and philosophy of mind, and are typically not used when discussing about adaptive software agents. Section 3 presents agent mobility and its relationship to adaptive agents. In Section 4, we discuss the relation between adaptive agents and agent communication and propose a model for enriching it. In Section 5, we introduce a way in which the internal behavior of an adaptive agent could be modeled and give a simple example of an adaptive mobile agent. Finally, Section 6 concludes the paper with a discussion about the future work.

2 Theoretical Background

This section presents three useful concepts related to our notion of adaptive agent: Gregorian creatures [4], scripts [11], and episodic memory [13]. These concepts are outside the core of software agent research. However, we feel that they can help to understand what we mean by adaptive agents. Additionally, our purpose is to give some practical guidelines for agent designers; implementing these three concepts can effectively help in creating an adaptive agent.

Gregorian creature is a concept invented by Dennett [4]. Dennett categorizes creatures into four main groups. These are Darwinian, Skinnerian, Popperian, and Gregorian creatures. They differ from each other in the ways they interact with the environment they are situated in. However, this division into four groups of creatures is not exclusive, but the groups are rather subsets of each other. Darwinian creature is the simplest; it has only one means of interacting with its surroundings. Creatures with better means survive better in the environment than the creatures that have poor methods. An example of Darwinian creature in the software agent world could be an agent that monitors its surroundings and responds the same way regardless of the type of stimula from the environment.

Skinnerian creature is a bit more complex than Darwinian. It has several methods of cooperating with the environment. When monitoring the surroundings, a Skinnerian software agent can respond to stimula in a number of ways. The fittest of these methods gets feedback from the environment causing the others to eliminate.

A yet more complex creature is the Popperian one. Unlike the two presented above, Popperian creature has an internal view of the world. This enables it to simulate the different means of interaction with the environment before trying them in practice. Popperian software agent that monitors its surroundings performs some internal processing and selection between the different ways to respond to a given stimulus before actually doing it. Finally, Gregorian creatures are the most intelligent. The difference between Gregorian creatures as opposed to the others is the capability of "externalizing the intelligence". This means that the Gregorian creature is able to store bits and pieces of its intelligence as mind-tools in its surroundings and later retrieve them for use. Having this capability, the Gregorian creature is the most important concept with respect to this paper.

A Gregorian agent monitoring its surroundings could process the input data to a more useful form and store this enriched data somewhere. Later on, for example, when the agent receives similar input data, it can bypass some or all of the potentially resource-consuming processing of the data by retrieving the stored and already enriched data for use. We design our adaptive agents so that they can upload domain-specific information to web servers and download it for use. These pieces of domain-specific information function as mind-tools for adaptive software agents.

A notion of script is useful when modeling environment-, task-, or domain-specific actions. Script is a concept invented by Schank [11] and it corresponds roughly to what we mean by domain-specific procedures. It is intended as a basic unit for modeling human memory. For example, going to local grocery store can be explicated with a script. This "grocery store visiting" script activates when

entering the store and is useless (maybe subconscious) at other times. Our attempt is to incorporate this notion into the world of adaptive agents. When entering a new environment, or if the present environment changes, the agent activates appropriate script, and acts accordingly.

Episodic memory [13], from human memory research, concerns personally experienced events whereas semantic memory means general world knowledge. In our model the agent may first download some domain-specific procedure from the server it is visiting, or possibly from some other server, then act according to the procedure, and at the end, possibly, upload the procedure back to the server. Some details of this procedure are changed based on the actions by agent. Next time the agent visits the same environment it considers the previous changes. Because the changes are personal to the agent, there are elements of episodic memory in this process.

Combining these three theoretical tools for a characterization of adaptive agent in web environment, we get the following: An adaptive agent is capable of uploading pieces of information as mind-tools to the web server and also downloading them for use (Gregorian creature). Mind-tools include not only facts, but also domain-specific actions (scripts). Details of mind-tools can be personally updated based on the behavior of the agent. These updates are taken into account subsequently when using them (episodic memory).

3 Mobility

Mobility is not conceptually necessary to agency. An agent that does not move is still an agent. Besides conceptually, mobility is also often bypassed when implementing agent systems. Using message passing can be more efficient than moving the agent in many cases (see e.g., [12]). Implementing mobility can however be justified if the agents are also adaptive. It might be appropriate to send an agent to a remote environment if it can adapt to local settings there.

Mobility in general can be divided into two categories: autonomous and involuntary mobility. Involuntary mobility can exist with whatever thing capable of moving. Planets orbiting in space give a good example of involuntary mobility. Autonomous mobility, on the other hand, presupposes an agent; there has to be somebody or something that decides to make the movement.

Mobility with respect to software agent applications can be divided into two main categories as well: mobile code and terminal mobility. Mobile code means that the agent transfers itself from one host into another. Moving the code and state near the data source can reduce communication overhead significantly, assuming the agent can perform some data filtering at the remote location. Terminal mobility occurs when the agent resides in a mobile device such as a handheld device or a laptop computer that is moved from one environment into another. For example, nowadays several wireless access technologies—such as GSM, GPRS, and UMTS in the near future—can be used to connect the mobile device to the fixed network. Should the mobile device change the access technology, the agents situated in that device may have to change their behavior to fit the chosen technology. There is no difference between mobile code and terminal mobility as far as adaptability is concerned. In both cases, the agent enters a new environment and has to adapt.

Mobile code is usually autonomous; the software agent moves itself according to its own decisions. The agent makes the final decision of its own movement even if the initial command comes from a human user or from another agent. Terminal mobility is typically involuntary with respect to the software agent. The mobile device is moved from one place into another in the physical world and the software agent typically has no control over it. However, in some cases, it might be the agent that initiates the terminal mobility. For example, the agent might be responsible for choosing the most suitable access technology. Now, the agent notices that a better access technology is available and performs (autonomously) necessary roaming activities. In this case, the terminal mobility can be considered autonomous from the agent's point of view. Although the terminal is not necessarily moving in this, the agent's environment might change dramatically.

Although there are many technical limitations and drawbacks with mobile agents, they might prove useful in some cases. If the task is very complicated, for example, mobile agents can prove to be an appropriate approach. An example of potentially complicated task is an auction that can include a number of messages sent between the auctioneer

agent and the bidders. Transporting the bidder agent to the auction environment might often seem more appropriate than sending all the bids over the network.

4 Communication

One of the key design rationale of software agents is decentralization. From implementation point of view agent-oriented programming can be seen as a follow-up or extension to object-oriented programming. So building a one-agent system corresponds to building an object-oriented system where all the functionality is coded in one class [15]. So the question arises: why design complex adaptive agents for different kinds of tasks, when you could accomplish the same with simpler and "stupider" task-specific agents? These simple agents need no adaptation skills, since they are designed for few predefined tasks.

The importance of adaptability can however be justified by looking at agent communication from a new angle. Traditionally agents communicate by sending each other messages that are processed the as soon as possible after they are received. In this traditional communication paradigm, the life span of the messages is quite short; they normally serve no function once they have been processed.

The notion of semantic web however enables a different way of thinking the communication. Semantic web technologies provide a way for an agent to publish its messages on web pages [3]. These messages can later on be indexed and then utilized either by the agent itself or by other agents. Should this publishing paradigm be adopted, there is no sense for agent to carry that information around. Not only would the amount of information be duplicated, but also synchronization problems could easily emerge. Obviously, this kind of agent communication is not going to replace the traditional agent communication, but could rather be used as an alternative in some cases.

The agent communicating like this should manifest some form of adaptability, however. For example, if an agent visits an auction environment on behalf of its master, it can download the auction-specific material from that environment. After the auction is complete, it can upload the material with possible updates back to the server and leave (cf. episodic memory). Such actions presuppose that the material in the auction environment is understandable to the agent. The agent works everywhere according to its general tasks, like the goal-formation from beliefs and desires. The specific tasks related to some environment follow these general tasks, but are used only in that particular environment.

5 Adaptability and Tasks

If agents are communicating based on the paradigm as presented above, it is important to decide what information is appropriate to store in the server. We do not take an exhaustive stand on this question, because it is quite case-specific. Sometimes storing the names or identifiers of the communicating agents is important. At other times, it is useful to store the things agents are communicating about and categorize them based on some ontology. However, one useful approach is to store the tasks specific to the domain agents are residing at or communicating about.

We feel that adapting to domain-specific behavior and procedures is at least as important as adapting to domain-specific facts. Therefore, we separate general tasks and domain-specific subtasks of agents from each other as depicted in Figure 1. The idea is that the agent does not carry the subtasks around with it. Adaptability is thereby needed from the agent to accommodate the subtasks with the general task it is executing. An example of a general task is buying a book. Possible subtasks for this would be paying for it and negotiating about the price.

Agent's general task is global; it remains the same regardless of the whereabouts of the agent.

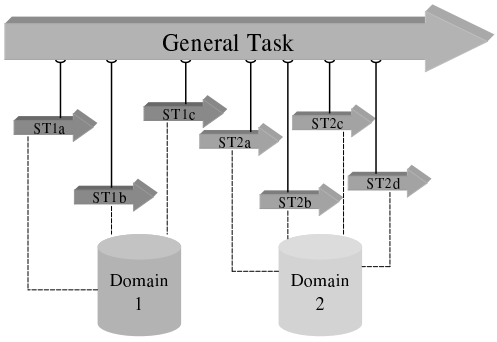

Fig. 1: General vs. specific tasks

Specific tasks, on the other hand, are local; they are used only within some domain. The domain-specific tasks are nevertheless dependent on the general task. They are reified as values of variables in the general task. Note that the agent can have more than one general task under execution at one time. As a simple example, let us assume that the agent have a goal for purchasing the book "Black and Blue" by Ian Rankin. Therefore, the general task of the agent could be defined as

GT="Buy the book 'BB' by IR"

The agent goes on pursuing that goal and visits different environments. Within each environment, the agent adapts to the domain-specific subtasks. Figure 2 depicts the agent's itinerary via several electronic bookstores with different ways of buying a book.

When arriving to the BOOKSTORE1, the agent downloads instructions how to use the service from the local server (1). The agent examines the instructions and finds out that BOOKSTORE1 provides only two services: The agent can query the price of the book and the agent can buy the book. The first one is carried out (say) using the fipa-query interaction protocol [5] and just giving the necessary information, for example credit card number, to the service, carries out the second one. Therefore, the agent creates and executes a domain specific subtask to find out the price of the book.

ST = "find out the local price using the fipa-query interaction protocol"

Let us assume, that the agent knows the fipa-query interaction protocol. Therefore, it can query the price immediately (2). Based on the price, the agent decides not to buy the book yet but visit other electronic bookstores first.

In the BOOKSTORE2, the agent again downloads the service usage instructions from the local server (3). Now, the agent finds out that the service provides a more sophisticated way of interaction, and that an interaction protocol called x-bstore2 should be used when communicating with the service. The agent does not know this interaction protocol, but the service description contains a pointer to another server, namely IP-SERVER, that contains the instructions for using this interaction protocol. The agent downloads the instructions from IP-SERVER

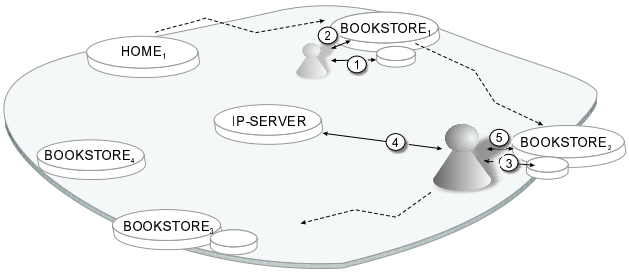

Fig. 2: Visiting different environments

(4), and then can interact with the BOOKSTORE2 server (5). After visited all the electronic bookstores, the agent decides to buy the book from some bookstore. The agent goes back to that server and gives the necessary information to the server and later the book is delivered to the owner of the agent.

The example above assumes that the agent can download service instructions from a local server and that it can understand those instructions. One possibility to achieve this is to encode the instructions using commonly agreed language. DAML+OIL [8] is a language proposed for describing things in the Semantic Web by defining ontologies. Domain-specific actions and facts could conveniently be coded as DAML ontologies. Subtasks are at the appropriate level for DAML encoding because they are used only within a particular domain. After entering the domain in question, agent downloads the appropriate subtask. The DAML encoding of the subtask is then translated into a form for the agent to utilize in its reasoning. After executing the subtask, the agent uploads it in DAML-format back to the server along with the changes based on this particular execution of the subtask in question.

6 Conclusions and Future Work

We presented a model of an adaptive software agent, which is capable of adjusting itself to different environments that may have features new to the agent. Furthermore, we introduced three useful concepts—Gregorian creatures, scripts, and episodic memory—related to our notion of adaptive agent. We also considered of implementing such agents in the Semantic Web environment. The future work includes further enriching the provided model.

Some questions related to our model are still unanswered. For example, an important thing is

to decide what information to store in web servers and in what detail. Should every bit of agent communication be stored, or just the abstract interaction protocols, for example? Also thinking the retrieval mechanisms for the stored content is important. Some ontology defining the properties for different interactions, communications, and other tasks would be useful for effective retrieval. The division between public and private information is also interesting. What information is retrievable for every agent, what is private for agents of some user, for example?

DAML-S is an ontology build on top of DAML+OIL for discovering, invoking, composing, and monitoring web resources offering particular services and having particular properties [8]. DAML-S consists of three specifications [1]. One contains an ontology for services, one for service profiles, and one for processes. We are planning to utilize the process ontology as an upper ontology for subtasks that the adaptive agents perform. FIPA (Foundation for Intelligent Physical Agents) interaction protocols could serve as examples of domain-specific procedures. We are planning to translate such interaction protocols in DAML and create an agent environment with agents capable of using it as described in this paper.

References:

[1] A. Ankolenkar et al. DAML-S: Semantic markup for web services. In *International Semantic Web Workshop — Infrastructure and Applications for the Semantic Web*, pages 411–430, Stanford, July 2001.

[2] E. Christensen et al. Web services description language (WSDL), 2001. Version 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[3] R. S. Cost et al. ITTALKS: A case student in how the semantic web helps. In *International Semantic Web Workshop — Infrastructure and Applications for the Semantic Web*, pages 477–494, Stanford, July 2001.

[4] D. C. Dennett. *Darwin's Dangerous Idea: Evolution and The Meaning of Life*. London: Allen Lane – The Penguin Press, 1995.

[5] Foundation for Intelligent Physical Agents. *FIPA Query Interaction Protocol Specification*. Geneva, Switzerland, Oct. 2000. Specification number XC00027.

[6] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proc. of the ECAI'96 Workshop on Agent Theories, Architectures, and Languages: Intelligent Agents III*, volume 1193 of *LNAI*, pages 21–36. Springer-Verlag, Aug. 1997.

[7] M. Gudgin et al., editors. *SOAP 1.2 Specification*. 2001. http://www.w3.org/TR/soap12-part1/, http://www.w3.org/TR/soap12-part2/. Work in progress.

[8] I. Horrocks, F. van Harmelen, and P. Patel-Schneider, editors. *DAML+OIL*. 2001. http://www.daml.org/2001/03/daml+oil-index.

[9] D. Kinny and M. P. Georgeff. Commitment and effectiveness of situated agents. In *Proc. of the Twelfth International Joint Conference on artificial Intelligence (IJCAI-91)*, pages 82–88, Sydney, Australia, Aug. 1991. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[10] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1995.

[11] R. C. Schank and R. P. Abelson. *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge*. Erlbaum, Hillsdale NJ, 1977.

[12] M. Straßer and M. Schwehm. A Performance Model for Mobile Agent Systems. In *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, pages 1132–1140, Las Vegas, NV, USA, 1997.

[13] E. Tulving. Episodic and semantic memory. In E. Tulving and W. Donaldson, editors, *Organization of Memory*, pages 381–403. Academic Press, New York, 1972.

[14] UDDI. The UDDI technical white paper, 2000. http://www.uddi.org/.

[15] M. Wooldridge and N. R. Jennings. Pitfalls of agent-oriented development. In *Proc. of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 385–391. ACM Press, May 1998.