Predictive Software Agents in Pervasive Computing

Mikko Mäkelä

Department of Computer Science, University of Helsinki

P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland

E-mail: `Mikko.Makela@helsinki.fi`

Abstract

We examine the possibilities for helping the development of multi-agent system containing learning agents targeted for predictive learning tasks. These kinds of agents are of use in realising seamless access to information anywhere and anytime, which is one of the central research areas for pervasive computing.

Contents

1 Introduction

Accessing and handling information anywhere and anytime when needed, seamlessly without the user having to learn complex technical skills is the goal of the new digital age. The first mass-produced devices targeted specially for this kind of *pervasive computing* have already appeared: AutoPC, Personal Digital Assistants (PDAs), Smart phones, wireless movable 'fridge appliances' with targeted UIs and Internet connections etc.

The requirement that information should be available anywhere means that wireless technology plays an important role in realising pervasive computing. Software agents are a relatively new programming paradigm, which shows much promise to tackle the problems in the field of distributed information management in general, and the problems which a wireless environment causes.

In this presentation we discuss developing learning agents, which are capable of carrying out predictions about future events and conditions in order to do planning and/or adapt to the changes in conditions. The focus is on helping the designing and implementation process of (learning) agent architecture for wireless environments. The emphasis is on establishing learning in a scalable and dynamic way, thus the ideas should be applicable to other resource scarce environments as well.

2 Software agents

There are numerous definitions for software agents. Some properties often attached to them are autonomity (the ability to operate by themselves), goal-orientedness (possibly including planning to reach the goal) and social ability (agents communicate with each other and the system).

It is easy to see that judging if a software entity has these properties is somewhat fuzzy (see e.g. [2] for a good overview): does for example a certain set of if-then-else-clauses make a software entity autonomous? The line between an object-oriented programming vs. agent approach has been found especially difficult to draw, even though there are clear differences. One of the most compelling is that objects do not have exhibit control over their behaviour, they must make methods available for other objects to invoke. Of course agents can be implemented using object-oriented techniques, but the point is that the standard object-oriented model does not include many of the aspects of agent programming (for a good discussion about this, see [3]).

In this presentation we are interested in *learning agents*, which try to adapt to the changes in their environment. From a machine learning point of view, passive modellers which merely produce an output from their inputs according to their internal mappings, can be seen as objects. The active modellers, whose set of outputs include actions which are meant to affect their environment, can be seen as agents. One way to formalise an agent is:

$$\text{Agent} = (P, S, \pi, A) \ ,$$

where P is the set of the agent's possible perceptions (inputs) p, A is the set of the agent's possible actions a, S is a set of possible agent's internal states s and π is the mapping producing A from S, usually called the *policy* of the agent. S may possibly be dependent of past states of the agent, meaning that agent may try to learn from its past experiences to make decisions about its future actions.

Because agents affect to their environment, they also affect to the inputs they will receive from their environment in the future. This cyclic dependence between agent's inputs (perceptions) and outputs (actions) means that we must use iterative methods for searching optimal policy of the agent. For example *Reinforcement Learning* [1] is often used for agents. In RL an agent explores to find the highest value of *utility* available according to its *utility-function.*

3 Motivating example

Why are predictive capabilities essential in realising pervasive computing and why should one implement them by using software agents? Because these questions are hard to answer in general terms, in this chapter it is illustrated with a little sample story, how predictions come in handy on many occasions and further, why a monolithic system does not come into question.

Sandra leaves her office for a meeting on the other side of town. Before she leaves, the files needed at the meeting are

> automatically synchronized between her laptop and desktop computers.
>
> In the local train Sandra listens to the weather forecast streamed via her laptop. Suddenly there is a slight change in the quality of voice, as the incoming email from her friend has large pictures attached. The email is about an on-going WWW-auction of a child's safety seat, just like the one Sandra was looking for. As the auction is closing soon, Sandra decides to make a bid immediately and switches her view to the web-browser, which has already logged into the WWW-pages of the auction-site.
>
> Because the system informs Sandra that the connection will likely drop soon, Sandra hurries and makes a bid for the safety-seat and approves the Shopping Agent to make a 10$ higher bid if neccessary. Just after the tunnel the connection is re-established, the streamed audio restarts and Sandra receives a confirmation that her bid was successful in the auction.

Let us examine how all functionality present in the sample story could be realized:

> Before Sandra leaves her office, the files needed at the meeting are automatically synchronized between her laptop and desktop computers.

Calendar Agent can predict Sandra's meeting, based on her input or e.g. because the meeting is frequent. It informs a Location Agent which has learnt to associate such information to the event of Sandra leaving the office, predicts that and passes the information to the File Agent. The File Agent predicts the files Sandra may need in the meeting and synchronizes them between laptop and desktop computers.

> While Sandra listens to the weather forecast, suddenly there is a sligth change in the quality of voice, as the incoming email from her friend has large pictures attached.

Although nothing special would appear to happen here, the email could be filtered to decide if it is important enough for connection establishment and possibly compressed before sending by a Compression Agent on the server side. Also, the fact that the voice quality just sligthly changes due to the email might be possible only if the compression ratio of the voice is dynamically adjusted *before* streaming blocks. This could also be done by Compression Agent which already knows about the email and thus about the need for adaptation.

> Sandra decides to make a bid immediately and switches her view to the web-browser, which has already logged into the WWW-pages of the auction-site.

The automatised logging into the WWW-pages of the auction-site could be realized by allowing e.g. the Keyword Agent to scan emails and find correlations about words and performed user actions. After learning the way to predict the fetching of WWW-pages, the Keyword Agent could pass the information to a Prefetch Agent. Because the fetching in this case involves logging, a Logging Agent which has user authorizations for different logging procedures migth also be involved.

> The system informs Sandra that the connection will likely drop soon.

The Location Agent predicts the incoming tunnel, possibly based on previous trips to the same meeting or information coming from the service provider, and informs the QoS Prediction Agent. The QoS Prediction Agent informs the user in the case of a likely connection drop. It might also make other preparations in order to minimize the damages resulting.

> Sandra makes a bid for the safety-seat and approves the Shopping Agent to make a 10$ higher bid if neccessary.

Here Sandra informs the Shopping Agent which then moves to the fixed network side to be her representative in the auction.

> Just after the tunnel the connection is re-established, the streamed audio restarts and Sandra receives a confirmation that her bid was successful in the auction.

This last paragraph gives our little story a happy ending, as agents' autonomity and internal error handling enable seamless recovery from the connection drop. Of course error recovery can be quite complex in reality, but it is out of the scope of this presentation.

It is worth noting that the explanations presented here are only drafts to illustrate agent approach to the problem and these solutions (used task decompositions etc.) are by no means the only possible ones.

While one could imagine a monolithic system to offer all this functionality, it is clear that no such system could scope with every possible example one could come up with just sligthly altering the described situation. Without deep modularity, the software would soon be too big to manage and too resource hungry for equipment used by nomadic users. Agents allow modular and dynamical implementation, where system's functionality is a result of relatively small and simple pieces of software (agents) co-operation. The fact that these software components are autonomous means that when the level of available resources is thigth, those agents which are not needed or can not perform satisfactorily at the time can be stored on persistent storage or even deleted without loosing the whole functionality of the system.

4 Learning Service

In this section we examine ways to support the agents in learning with easy to use and dynamically initiated modellers.

4.1 Modeler agents

Historically, the research on artificial intelligence can be divided roughly into two sectors: the expert knowledge approach and the machine learning approach. These two viewpoints on intelligent behaviour are both very important for an agent system where adaptation is neccessary, but on the other hand can not take too much time or be too random a process. In such a situation, we need learning to make the adaptation possible, and to guide and speed up the learning we need expert knowledge of the learning domain.

The expert knowledge is typically represented as either hard-wired code in an agent, symbolic representations allowing *symbolic reasoning*, or both. We call agents possessing expert knowledge *intelligent agents*. Their properties are not considered to include learning or adaptation to the environment, which is what separates so called *learning agents* from them.

The difference between intelligent and learning agents is not in reality always so clear. But this distinction serves our purposes here, as it illustrates a clear way of making a learning agent out of an intelligent agent by adding adaptive capabilities to it. This can be done by offering help for the modelling of the world in which the intelligent agent is.

We propose a Learning Service, which offers the possibility for agents to easily initiate different kinds of modellers in order to use them as support for decision making. These models can be equipped with agent properties in order to make them interoperate easily and have the basic means for error handling etc.

We call these agents which are initiated by the Learning Service (based on the requests from intelligent agents) as Modeller Agents. We see several potential benefits in using them:

- They facilitate the use of modelling in agent systems by offering an easy-to-use interface to a set of generic model types.
- The initiation and change of different kinds of models is dynamic, which offers good grounds for developing scalable solutions, where environmental conditions affect the modelling methods chocen.
- The dynamical life span of Modeller Agents also makes it possible to try making automated searches for the best available modelling method. This could come in handy especially during the development of the system, although a large set of usable algorithms and common ways to guide the search among them would most probably be needed to make this approach really useful.
- Using the same algorithms for many agents has the potential to save memory within most environments, as the code does not have to be reproduced.

As with any service, the careless usage of the Learning Service has also a potential to waste resources. For example in ready-for-market systems the initiation of the modellers should surely be based on known needs, especially if the resources are limited, which is the case most often within pervasive computing.

5 Perception Service

In this section we examine what benefits could be reached by centralized handling of agents' perceptions and propose a new agent service for the task.

5.1 Adaptation via Chains of Interaction

When humans or animals react to the changes in environment, the final action is a product of a series of events. First, the change activates a perception within the sense organ affected. If the phenomenon causing the perception in question happens to be e.g. a voice, after arrival of the air pressure changes in the ear, raw information of them is then processed step by step in different parts of the brain. When finally the perception is brougth to the locus of attention, it might already be associated with meanings and mental images, which the voice has triggered. However, not all perceptions come to the locus of attention, and not all of those which do, generate any physical action.

The framework presented here can be seen having resemblence to this cultivation of perceptions descibed above. Perceptions are caused by the environment and by the agents. The system is not restricted to "real-time" perceptions. Different kinds of models can be used to cultivate perceptions (clustering, filtering etc.) and bring memory properties to the system. In most multi-agent environments the cultivation and memorisation of perceptions would occur only in each agent separately.

The cultivation of perceptions happens in chains of agent interoperation, where the agent preceding in the chain is *perception producer* for the following agent, and the following is *perception consumer* for this interoperation. One form of cultivation is modelling, where those aspects of the information which are seen as critical for decision making are extracted.

In the former section we introduced one way for adding modelling capabilities to the system with the help of the Learning Service. In this section we are interested in finding ways to make the information flows between agents more easily administret.

5.2 Centralized handling of perceptions

In the story, we did not yet deal with the problem of how the agents find each other in order to interact and co-operate. This is in fact one of the basic problems for any multi-agent system, where all agents can not be known at the time of implementation. The usual solution to this problem is to use some agents, which all agents know, as brokers. Within this approach the agents are usually divided into three types: broker, service and client agents. Client agents use the services provided by service agents in order to accomplish tasks they cannot solve on their own.

While this approach to agent interaction is very general, it does not offer us much help for the central problem in learning to predict future events: finding correlations between different situations and events. This means that usually the correlation must be known to exist at the time the agent is implemented, or otherwise it may never be found.

To tackle for example this problem we propose a Perception Service, to which agents may offer all perceptions they think migth be off value for learning in the system. We see that there appears to be many reasons why a single entity handling perceptions in a centralized way could be of value:

- The scalability of the system. The centralized handling of perceptions is important for the scalability of the system, as it makes it simpler to use common models within Perception Service for adjusting the amount of storing the perceptions, mapping them to less resource-consuming structures etc.

- Easy organization of hierarchical learning. The above-mentioned mappings can be, for example, clustering, complex models for predicting future values of given perceptions, or anything in between. So besides scalability, centralizing of perceptions is good for efficient and more easily organised learning. Taking the Modeller Agents into account is easy by supporting e.g. periodic *subscriptions* to the information the models need for updating.

- Finding unknown correlations. If the handling of perceptions were decentralized, some dependencies could be impossible to find. Because all learning tasks are not known beforehand, there is not even in principle a way to group perceptions to isolated groups without the possibility that an optimal learning model would need perceptions from different groups. If the handling of perceptions is centralized, it is easy to collect data even for datamining purposes, when the resources allow. This can be really valuable in complex domains.

- No sacrifice of agent autonomy. As the name perception *service* suggests, the centralization does not restrict the agents' behaviour, it can be considered as an optional service for them.

We see that learning for adaptation will in most cases be hierarchical such way that 'cultivative' information flow between agents is one directional. This is the case for example in all the outlined chains of interaction of the example story in section 3. This kind of hierarchical agent interaction bears similarity to so called Layered Learning, which was developed by Stone to handle complex multi-agent domains. It has been very succesfully deployed in e.g. robotic soccer [4].

One very important aspect of the Perception Service is that it gives us an easy way to use much more resources on system development time for learning than will be possible to give when the system is ready. The developers can for example use more memory than is usually available for end-users in order to perform datamining for finding correlations.

6 Optimising the system

If we can construct a utility-function to calculate the overall utility the agent system receives, we can see the agent system as a whole as one agent and sometimes even try to use the same methods as for individual agents for optimisation.

For example in a setup consisting of a user's wireless terminal and a node representing the fixed network the terminal is connected to, the overall utility-function consists of at least the following elements:

- User Rewards (UR), like the reward of the user getting data without too much inconvience or delay etc.
- User Costs (UC), like the cost of data transfer etc.
- Service Provider Costs (SPC), like the cost of prosessing data on the fixed network side etc.

Thus the overall utility-function can be expressed:

$$U_g = \text{UR} - \text{UC} - \text{SPC} \ .$$

All of these utilities (rewards and costs) can be composed of a set of lower level utilities. The fundamental problem for using the utilities for system optimisation is nevertheless that there are no general ways to balance them well. For example, to balance the user rewards and costs concerning data transfers, we would need to know how much the user is willing to pay for certain data to be transferred in a certain time—in certain conditions. It is obvious that regardless of how advanced our user modelling methods are, we can never know exactly how the user feels at the moment about given data, without her telling it via the user interface somehow.

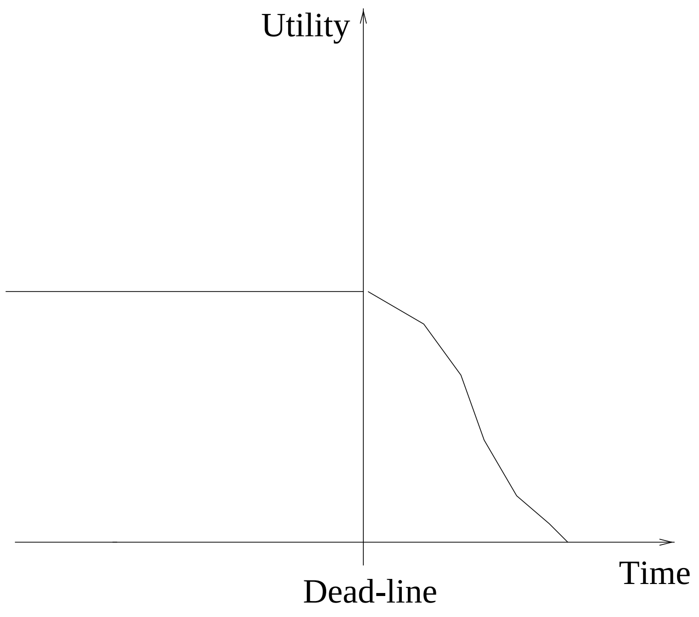

Figure 1: *Example of utility-function for receiving the data*

However, for optimisation purposes *during the development time of the system*, we can use models of different kinds of users. In the case presented above, we for example know the approximate form of the utility-function for the user receiving the data (Figure 1). To study the domain field, we can run lengthy simulation runs with different kinds of utility-functions and calculate an overall utility of different solutions.

If there are not fundamental flaws in simulations or utility-functions, the solution giving the higher overall utility over time should in the majority of cases be a better solution in a real-life situation than the one with lower overall utility. Of course the limits of simulations and constructed utility-functions must be taken into account, therefore expert knowledge of the domain should be used when evaluating the solutions. While the results we get this way are might be only indicative, they can still be very valuable in comparing one solution over another and finding possible bottle-necks of the system etc.

7 Conclusions

In the area of pervasive computing, where seamless user experience and the 'vanishing' of underlying computing are goals, predictions about the future conditions and events are of great importance. In this presentation we examined ways to make it easier to develop collaborative learning agents which have predictive tasks.

In the Monads project, the central ideas of the presented framework, the Perception Service and the Learning Service, have been implemented using Java. They have been used extensively in realising a multi-agent system where agents perform predictive tasks, essentially the prediction of Quality of Service (QoS) the user is about to face in her near future. The evaluation is performed in a simulated wireless environment, where simulated nomadic users perform different kinds of data transfers. While the evaluation of the system is still unfinished, we believe from our experiences in realising the predictions present at the moment, that the approach bears potential and deserves further research.

A very interesting subject for further research would be to examine the possibilities for finding a good way to give boundaries to the learning process running in simulation and how the search for the best solution inside these boundaries could be easily guided.

The points taken in this presentation can be summarised:

- The function of a multi-agent system where aim is in hierarchical learning can be seen as information flows inside chains of agents.
- These flows may cross and mix, the goal is to cultivate the information for decision making.

- In complex domain information flows can be realised in myriads of ways and there are many open problems in deciding, which of these is the one giving the optimal solution. Thus ad hoc methods and iterative trial & error approaches are used.
- This means that there is a need to make trialing between different solutions as easy as possible.
- We propose two new services: the Learning Service for realising a dynamic and easy way to use generic algorithms for modelling purposes, and the Perception Service for making the realisation of information flows from one agent to another easier, especially in the case of so-called Modeller Agents initiated using the Learning Service.
- We see the use of these services as complementary to existing agent frameworks, their use should not restrict the use of other methods.
- For comparing different solutions, an overall utility-function of the system should be constructed, if possible. It can be used to examine the system with simulations of real-life situations.

References

[1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, *Reinforcement Learning: A Survey.* Journal of Artificial Intelligence Research 4, 1996. pp. 237–285.

[2] S. Franklin and A. Graesser, *Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents.* Proceedings of the (ECAI)'96 Workshop on Agent Theories, Architectures, and Languages: Intelligent Agents (III). 1997. Springer.

[3] N. R. Jennings, K. Sycara, and M. Wooldridge, *A Roadmap of Agent Research and Development.* Journal of Autonomous Agents and Multi-Agent Systems, vol. 1, no 1, 1998. pp. 7–38.

[4] P. Stone, *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer.* MIT Press, 2000.