Reinforcement Learning of User Preferences for a Ubiquitous Personal Assistant

Sofia Zaidenberg, Patrick Reignier

Reinforcement Learning of User Preferences for a Ubiquitous Personal Assistant

Sofia Zaidenberg and Patrick Reignier
Prima, Grenoble Informatics Laboratory (LIG) / INRIA
France

1. Introduction

New technologies bring a multiplicity of new possibilities for users to work with computers. Not only are spaces more and more equipped with stationary computers or notebooks, but more and more users carry mobile devices with them (smart-phones, personal digital assistants, etc.). Ubiquitous computing aims at creating smart environments where devices are dynamically linked in order to provide new services to users and new human-machine interaction possibilities. *The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it* (Weiser, 1991). This network of devices must perceive the context in order to understand and anticipate the user's needs. Devices should be able to execute actions that help the user to fulfill his goal or that simply accommodate him. Actions depend on the user's context and, in particular, on the situation within the context. The context is represented by a graph of situations (Crowley et al., 2002). This graph and the associated actions reflect the user's work habits. Therefore it should be specified by the user him-self. However, this is a complex and fastidious task.

The objective of this work is to construct automatically a context model by applying reinforcement learning techniques. Rewards are given by the user when expressing his degree of satisfaction towards actions proposed by the system. A default context model is used from the beginning in order to have a consistent initial behavior. This model is then adapted to each particular user in a way that maximizes the user's satisfaction towards the system's actions. The ambient intelligence application domain imposes constraints that we had to consider for the selected reinforcement learning approach.

We will first introduce the Ambient Intelligence application domain and the associated constraints. We will then present a qualitative user study conducted to validate our hypothesis. We will motivate our reinforcement learning algorithm selection and present the way we have modified it to fulfill our particular constraints. We will then present experimental results.

2. Pro-active Ambient Intelligence

Our research domain is Pro-active Ambient Intelligence applications development. We have decided to use reinforcement learning as a paradigm to let the end user adapt the application's behavior to his own needs. In this section, we will briefly present the Ambient Intelligence domain to motivate this choice and to introduce the particular constraints we have to manage.

2.1 Ubiquitous computing: Weiser's vision

In the late 1980s and early 1990s, Xerox PARC researcher Mark Weiser developed the concept of ubiquitous computing presented in his seminal paper: The Computer for the 21st Century (Weiser, 1991). He characterized the computer evolution in three main eras:

1. mainframes: a central processing unit shared by a group of users,
2. personal computers: one central unit per user,
3. mobility: several processing units per user, following his movements.

The integration of computing devices into everyday environments has been one of the predominant trends over the last decade. Cell phones, PDAs and laptop computers as well as WLAN networks have become part of almost every household. This trend enables computer-everywhere environments. The objective is to make computers not only user-friendly but also invisible to the user. Interaction with them should be possible in forms that people are naturally comfortable with:

> *"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."*

Some of the first ubiquitous applications were the *tabs*, *pads* and *boards* developed by Xerox PARC between 1988 and 1994 (Adams et al., 1993). Other examples can be found from the *Things That Think*[1] consortium of the MIT Media Lab. This group is inventing tomorrow's artifacts by embedding computers in everyday life objects.

In 1995, Weiser has introduced the new notion of calm computing. Calm computing is an approach that engages both the center and the periphery of our attention. Our attention can move from one to another. When we drive a car, our center of attention is on the road and the noise of the engine is on the periphery. If this noise is unusual, our attention will instantly move from the center to the periphery. Periphery can inform without overwhelming, center allows to get control.

The multiplication of computing devices goes also with a proliferation of interconnected sensors. These sensors can measure physical parameters (temperature, humidity, light, etc.). They can also be software probes as for instance the next appointment in his diary, the arrival of a new email. Ambient Intelligence (or AmI) is the conjunction of ubiquitous computing and artificial intelligence. The goal is to exploit the perception capacities of all these sensors to analyze the environment, users and activities and allow the system to react to the *current context*. Ambient Intelligence concept has been first defined in 1998 by Philips in their reflection on the future of consumer electronic equipments. Among the very first applications, we can cite the Coen Intelligent Room (Coen, 1998), the Abowd eClass Project (Abowd et al., 1996) or the Mozer Adaptive House (Michaël, 1998).

In the rest of this chapter, we will consider *pro-active* Ambient Intelligence applications, as defined by Salovaara and Oulasvirta (Salovaara & Oulasvirta, 2004):

> *"... the concept proactive refers to two critical features of a system: 1) that the system is working on behalf of (or pro) the user, and 2) is taking initiative autonomously, without user's explicit command."*

[1] `http://ttt.media.mit.edu`

2.2 Where are we now?

Since 2000, the number of computer devices in our personal and professional life has exploded: personal computers, smart-phones, PDAs, video game consoles connected to the Internet etc. Weiser's vision of ubiquitous computing has been partly achieved (the multiplication of CPUs around us). But pro-active Ambient Intelligence applications are still in the laboratories and did not reach everyday life. What are the difficulties?

There are many. Some of them are directly inherited from "classical" Artificial Intelligence, as for instance the *frame problem*: how should we model the environment (context model) and how shall we update this model based on the sensors' values (Lueg, 2002)? In this chapter, we will focus on the important problem of the user's confidence in his proactive applications.

User's confidence in an automated system is a common problem that is not limited to Ambient Intelligence. In particular, Muir and Morray have shown in the field of process automation that the user's trust in an automatic system is directly related to the user's perception of the skill of that system (Muir & Moray, 1996). The end user should not underestimate or overestimate the capabilities of that system to optimally use it (calibration of trust).

One way to establish this trust relation is to let the system expose its internal behavior to the end user. This is what Bellotti and Edwards named the *intelligibility* (Bellotti & Edwards, 2001):

> *"Context-aware systems that seek to act upon what they infer about the context must be able to represent to their users what they know, how they know it, and what they are doing about it."*

Cheverst (Cheverst et al., 2005) talked about *comprehensibility* to suggest that the user should be able to look inside the device (like a glass box) to examine its inner working. In particular, comprehensibility reduces the fear of having a system doing something "in our back" (Abowd & Mynatt, 2000). Comprehensibility is also associated to *scrutability*, which refers to the ability for a user to interrogate his user model to understand the system's behavior. As stated by Kay (Kay et al., 2003), scrutability is contradictory with the invisible computer concept as defined by Weiser, but it seems to be necessary to gain the user's acceptance.

Based on scrutability and on the level of system control versus user control (pro-activity), Cheverst is categorizing the various Ambient Intelligence applications. In figure 1 adapted from his article, the gray circle characterizes the kind of applications we want to develop.

Developing a pro-active Ambient Intelligence application is a complex task. It cannot be supported only by the developer or the end user. Pattie Maes argued that to gain trust, the end user must be involved in the specification of the system's behavior, but he usually cannot directly program it (Maes, 1994). User's habits are also evolving through time (Byun & Cheverst, 2001), implying repeated modifications of the application. Machine learning has been proposed as a possible solution for those problems. In particular, Remagnino (Remagnino & Foresti, 2005) thinks that the future of Ambient Intelligence is correlated to machine learning researches because associations between sensory outputs and intelligent behavior are too complex to be hand-coded.

We have presented in this section a brief introduction on Ambient Computing. We have focused on particular characteristics that those applications must exhibit to gain user's acceptance. This gives us some constraints that must be considered for building our assistant. To validate those constraints, we have first conducted a user study that we will present in the next section.

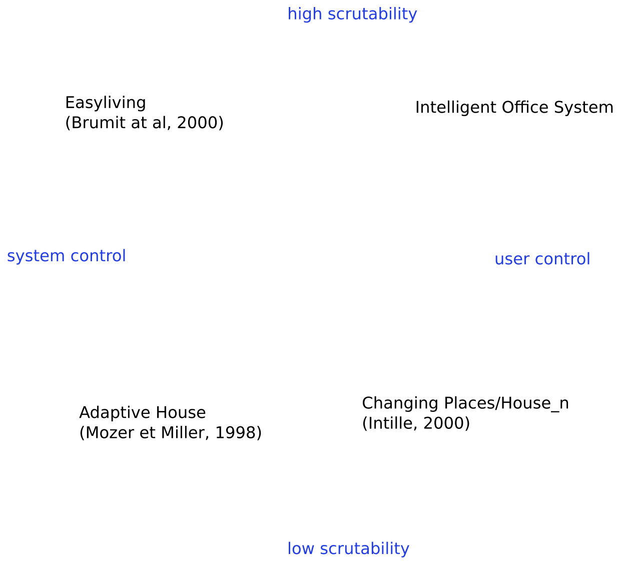


Fig. 1. Two-dimensional design space spanning control and scrutability dimensions, adapted from (Cheverst et al., 2005): the gray circle characterizes the applications we are considering.

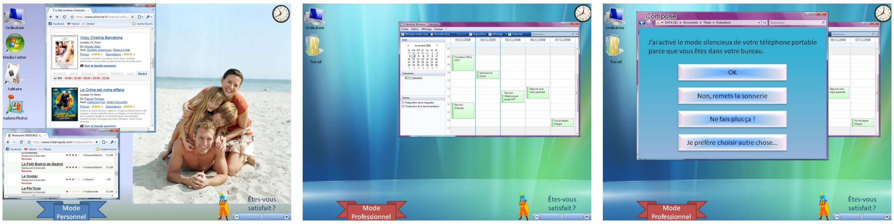
3. User study

The goal of this user study was to measure the expectations and needs of users with regard to an ambient personal assistant. Subjects were 26 active persons, 12 women and 14 men, distributed in age categories as follows: 9 subjects between 18 and 25, 7 between 26 and 40, 7 between 40 and 60, and 3 over 60. None of the subjects had advanced knowledge in computer science.

3.1 Description

The study was based on ∼1 hour interviews with every subject. The interviewer followed a predefined script. The script started with a few open questions about information and communication technologies to evaluate the subject's general knowledge, but also his perception and his uses of such technologies. Then, the interviewer presented our ubiquitous system using a model (an interactive power point presentation: some of the slides are shown figure 2). This interacting powerpoint was exposing a simple scenario about the user's laptop. The scenario starts in the morning and the user is at home, browsing for movies and restaurants (figure 2(a)). When he arrives at work, the laptop automatically switches to the user's "work" setting (figure 2(b)). Then, the assistant turns the user's cellphone to vibrate and displays a message about this action. The user can ask for an explanation about this action and choose to undo it or select another action for this situation (figure 2(c)). At the end of the day, the system switched back to the "home" setting. The interviewer explained also orally other examples of services that could be offered by the assistant.

After the presentation, the subject was asked for his opinion about such a system. He could freely express the advantages and drawbacks of what he saw and the situations in which he thought the assistant was particularly useful or interesting. This gave him the opportunity to


J'ai activé le mode silencieux de votre téléphone portable parce que vous êtes dans votre bureau.

(a) Slide 1 (b) Slide 2 (c) Slide 5

Fig. 2. A few slides from the model used to present our system to the subjects.

talk about ubiquitous assistants in general and about what their usage implies for his everyday life. Another goal of the conversation was to determine the acceptability of the system. The interviewer asked the following questions:

- "If the assistant learns badly, if it offers you wrong services at wrong times, what would be your reaction?"
- "If the assistant makes mistakes, but you know that he is learning to adapt to your behavior, would you give him a chance?"
- "Would you accept to spend some time to answer questions to make the assistant learn more quickly?"
- "What would you gain from getting an explanation about the assistant's decisions?"

We were also interested in finding out if the subjects would feel observed and fear that their privacy was in jeopardy. If they would not bring the subject up themselves, we would ask questions about this.

3.2 Results

After analyzing all the interviews, it appeared that 44% of subjects were interested in our assistant, and 13% were conquered. Interested persons share the same profile: they are very active, very busy in their professional as well as personal lives, they suffer from cognitive overload and would appreciate some help to organize their schedule. Other noticeable observations standing out from the interviews are the following:

- Having a *learning* assistant is considered as a plus by users. In fact, subjects felt a learning system would be more reliable since it would respond to their own training.
- Users prefer a gradual training versus a heavy configuration at the beginning.
- This training must indeed be simple and pleasant ("one click").
- The initial learning phase must be short (one to three weeks).
- It is absolutely necessary for the assistant to be able to explain its decisions. This aspect was particularly discussed by (Bellotti & Edwards, 2001).
- The amount of interactions wanted between the user and the assistant varies from one subject to another. Some accept only to give one-click rewards while others would be happy to give more inputs to the system. This confirms that people are interested in *engaging* systems, as stated by (Rogers, 2006). For those users, we could add an optional

debriefing phase where the assistant goes through the learned behavior and the user corrects or approves it.

- Mistakes made by the system are accepted to some extent as long as the user knows that the system is learning and as the system is useful enough to the user. But errors must not have critical consequences. Users always want to remain in control, to have the last word over the system and even have a "red button" to stop the whole system at any time.
- Some subjects pointed out that the assistant could even reveal to them their own automatic and subconscious customs.
- A recurrent worry expressed by interviewees is the fear of becoming dependant of a system that cares for them and becoming unable of living without it (what if the system is broken-down?).

This user study justifies our ubiquitous assistant since a sizeable part of interviewed subjects were prone to using it. Points listed above give us constraints to respect in our system. They will be listed in the next paragraph.

3.3 Our constraints

Based on the Ambient Intelligence presentation (section 2) and this user study, we will now present the constraints we have considered for our approach.

We want to build a personal assistant whose behavior is learned from user inputs. We have to respect several constraints:

(a) The system must not be a black box. As detailed in (Bellotti & Edwards, 2001), a context-aware system can not pretend to understand all of the user's context, thus it must be responsible about its limitations. It must be able to explain to the user what it knows, how it knows it, and what it is doing about it. The user will trust the assistant (even if it fails) if he can understand its internal functioning.

(b) The training is going to be performed by the user thus it must be simple, non intrusive and it must not put a burden on the user.

(c) The learning should be *Life-Long* learning to continuously track the user's changes of preferences.

(d) The training period must be short, unless the user changes preferences.

(e) The system must have an initial behavior that is not incoherent.

We have been exploring in a previous work a supervised learning approach for situation ↔ action association (Brdiczka, 2007) for a virtual assistant. Another example is the *a CAPpella* system (Dey et al., 2004). Even if they produce some promising results, both approaches are based on an off-line annotation of recorded data. This annotation process can be quite painful for the end user and because of the off-line nature of the process, it is performed "out of context". To overcome these limitations, we have been considering Reinforcement Learning as a solution for getting "in context" qualitative (not too intrusive) feedback from the user.

4. Reinforcement learning: a quick overview

4.1 Definition

Reinforcement learning (Sutton, 1988; Sutton & Barto, 1998) is an approach where an agent acts in an environment and learns from its previous experiences to maximize the sum of rewards received from its action selection. The reward is classically a continuous function between -1 and 1. 1 corresponds to satisfaction, -1 disapproval and 0 means no opinion.

In reinforcement learning approaches, the environment is typically modeled as a Markov Decision Process (or MDP). A Markov Decision Process is defined by $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$:

- $\mathcal{S}$ is the set of all possible states of the environment.
- $\mathcal{A}$ is the list of possible actions.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [-1; 1]$ is the reward function. $\mathcal{R}(s,a,s')$ indicates the opportunity to choose action a in situation s.
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0; 1]$ is the transition function modeling the environment. $\mathcal{P}(s,a,s')$ gives the probability of being in situation s' when applying action a in situation s. The Markov property specifies that the next situation depends *only* on the current situation and action and is not based on previous situations.

With Markov Decision Processes (and so, with classical reinforcement learning approaches), the current state must be completely known. The environment's evolution may be stochastic (hence the probabilistic transition function), but it must be *stationary*: the probabilistic law must not change over time.

An agent has a policy function $\pi : \mathcal{S} \to \mathcal{A}$, proposing an action for every situation: the agent's behavior. This policy function can be evaluated using a *value function* $V^{\pi}(s)$. This value function computes the expected discounted future rewards received by the agent being currently in state s and applying its policy π. This value function is expressed by the following equation:

$$V^{\pi}(s) = E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\} \text{ with } 0 < \gamma < 1 \tag{1}$$

The goal is to build an optimal policy π^* maximizing the corresponding value function V^{π^*}. This optimal value function is the solution of the recursive Bellman's equation (equation 3):

$$\begin{aligned} V^{\pi^*}(s) &= max_a \left(E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\} \right) && (2) \\ &= max_a \left(\sum_{s'} \mathcal{P}(s,a,s')(\mathcal{R}(s,a,s') + \gamma V^{\pi^*}(s')) \right) && (3) \end{aligned}$$

Once the optimal value function is determined, the optimal policy is directly calculated with:

$$\pi^*(s) = argmax_a \left(\sum_{s'} \mathcal{P}(s,a,s')(\mathcal{R}(s,a,s') + \gamma V^*(s')) \right) \tag{4}$$

If we have a model of the environment (the $\mathcal{P}$ and $\mathcal{R}$ functions of the corresponding Markov Decision Process), the Bellman equation is fully determined. The optimal strategy can be found using dynamic programming. It is an off-line optimization problem.

If the environment model is unknown, there are two main categories of approaches:

Model-Free : no explicit model of the environment is constructed. The optimal strategy is built as a result of the agent's interactions with its environment. Some examples of Model-Free approaches are: Q-Learning (Watkins & Dayan, 1992), SARSA (Rummery & Niranjan, 1994), Actor-Critic etc.

Model-Based : a model of the environment is constructed (learning phase) and then exploited (planning phase) to build the optimal strategy. This is for instance the Dyna approach (Sutton, 1991) that we will present in subsection 4.4.

4.2 Examples

Reinforcement learning application domain covers a lot of ground, from robotics to industrial manufacturing or combinatorial search problems such as computer game playing. We will present some examples involving pro-active agents interacting with an end-user.

Robyn Kozierok and Pattie Maes (Kozierok & Maes, 1993) have proposed in 1993 an agent helping the user to schedule meetings (agenda assistant). One of the very interesting aspects of this approach is the smooth control transfer from the end user to the agent. This smooth transition allows a progressive trust relation establishment, an important step for the system acceptance. The agent's learning is a combination of memory based and reinforcement learning. The agent is first observing the end user, recording every *situation* ↔ *action* association. When a new situation is detected, the agent extracts from its database the closest previously observed situation and the associated action. If the proposed action is incorrect, reinforcement learning is used to try to correct it. This approach is very interesting because it combines two important properties: establishing a trust relationship with the user based on a smooth control transfer and consolidating this relationship with an intelligible model. If the agent is choosing a wrong action, the system tells the user the reasons for its choice and the user can explain why this is incorrect. One of its main drawbacks is the simplicity of the situation model (well suited for the agenda problem).

Another example is Walker's application of reinforcement learning to dialog selection in a spoken dialog system for emails (Walker, 2000). The agent must learn to optimally vocally interact with the end user to quickly provide him the right information. The difficulty of this application is the complexity of the state space: 13 discrete variables that can take between 2 and 4 different values each. The authors recommend to reduce the state space to significantly improve the system's performances. This state space reduction is done by searching for irrelevant variables in the learning process.

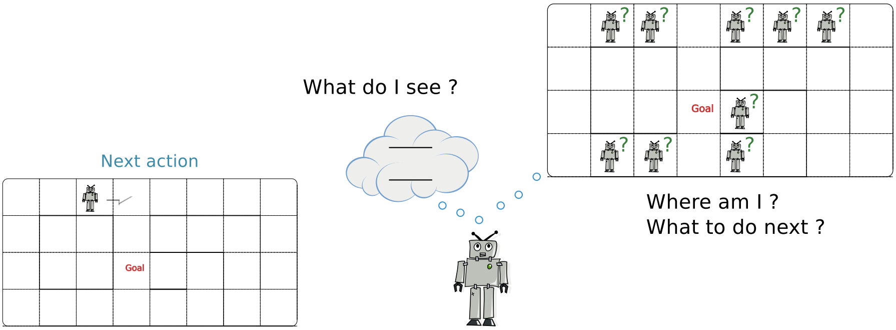
4.3 The Markovian hypothesis

Our learning agent perceives the environment's state. This environment is both "physical" (user entering a room, . . .) and "computer based" (new email arriving, agenda alarm, . . .).

In our problem, the agent is not the only acting entity modifying the environment. The environment is modified by external elements, out of the agent's control. The end user is one of those elements: for instance, he enters or leaves his office, triggering localization events, sends emails, etc. All those actions modify the environment. They are motivated by the user's internal state. The agent would need to access this internal state to fully understand and predict the environment's evolution. As this is not possible, the agent has only a *partial* perception of the environment.

4.3.1 Partial environment perception

An agent in a Markov Decision Problem that has only a partial perception of its environment breaks the Markov hypothesis. Let us consider for instance the classical maze problem. If the robot has an exact perception of its state in the environment (position and nearest obstacles), it might be able to correctly decide its next action to reach the goal (see left part of figure 3). If it has a partial perception (the nearest obstacles for instance), there is an ambiguity on its real position: it cannot decide anymore what to do based on the current perceived state (see right part of figure 3). The Markov hypothesis is no more true: it needs to remember previous states and actions to disambiguate its current position. The action selection is no more based solely on the current state.



(a) The robot has a complete perception of its state

(b) The robot perceives only surrounding obstacles

Fig. 3. Consequences of having a complete or partial state observation

When an agent has only a partial access to the current state, Markov Decision Processes are then replaced by Partially Observable Markov Decision Processes[2] (Aström, 1965). In a Partially Observable Markov Decision Process approach, the agent does not know its real state s but it only has access to an observation o. It then acts based on an estimated state b (belief state) defined by a probability distribution on S (all states). A Partially Observable Markov Decision Process is a non-Markovian process which can be reduced to a Markovian process on a continuous state space: the belief space. The belief space is the space which, from the probability of being in each state and an action, gives the new probability of being in each state (for a quick introduction, see for example (*POMDPs for Dummies: Page 5*, 1999)).

Partially Observable Markov Decision Processes are used in particular for multi-agent problems. In a multi-agent system, each agent has access to its internal state, to the external environment state but has no perception of the other agents' internal state: each agent has a partial view of the global system's state (Littman, 1994). If each agent is governed by a Partially Observable Markov Decision Process, we talk of Decentralized Partially Observable Markov Decision Process or DEC-POMDP.

Solving a Partially Observable Markov Decision Process problem is *p-space hard* (Papadimitriou & Tsitsiklis, 1987). There are some approximate solutions: (Pineau et al., 2003), (Spaan & Vlassis, 2005) or (Smith & Simmons, 2005) for instance. But it remains very difficult, especially with convergence speed constraints imposed by the end user being in the loop.

[2] POMDP

4.3.2 Non Markovian stationary or Markovian non stationary?

Considering the end-user as part of the environment, our learning agent problem can be naturally modeled as a Partially Observable Markov Decision Process. The agent cannot perceive the end-user's internal state, responsible for the user's actions and part of the environment evolution. As stated by Buffet in his PhD (Buffet, 2003), our system is non Markovian and stationary. It is stationary because:

- the environment without the user can be considered as stationary (the rules of evolution are not changing);
- the end user might introduce a non stationary aspect (his behavior is evolving through time) but this non stationary part is embedded in the non observable part.

We could also consider that the end-user is not part of the agent's environment. The agent has a full perception of the state: the problem is now Markovian but it is no more stationary. The end user is a "disruptive" element, causing non deterministic environment state changes. As we have seen in subsection 4.1, a non deterministic evolution of the environment is compatible with a Markov Decision Process as long as the probabilistic evolution law is not changing (stationary). In our case, the non stationary part corresponds to the end-user's behavior evolution. We can consider that this behavior evolution is slow and that our problem is locally stationary. If the agent's learning speed is much faster that the user behavior's evolution, then the agent can track user evolutions and constantly adapt to them.

As we have explained in the previous section, Partially Observable Markov Decision Process approaches are difficult. We have preferred to use a more classical Markov Decision Process approach, selecting the second solution: the end user is seen as a "disruptive" element, outside of the agent's environment.

4.4 Reducing user's burden: indirect reinforcement learning

Reinforcement learning approaches need a lot of steps to converge. For instance, the backgammon agent had to play 1.5 million games to learn (Kaelbling, 2004). In a reinforcement learning approach, the agent is building its policy through interaction with its environment. If this environment is virtual (like for the Backgammon game), the convergence time (due to a high number of learning steps) can be partly reduced by parallelizing the algorithm for instance or by using faster processors. But in a real environment, especially if the end-user is part of the learning step giving for instance the rewards, the bottleneck is not the processor speed but the end-user himself who can quickly get bored.

To limit end-user's interactions, indirect reinforcement learns a real world model that can be used by the agent as a "virtual playground" to produce as many off-line experiments as necessary. The world model allows *imaginary* experiments as defined by Craik (Craik, 1943). The world model is a transformation $S \times A \to S$. Building a world model is a life-long learning process. The agent is repeatedly updating its world model based on new available observations (s, a, s') coming from the real environment (supervised learning). Because it is a life long process, the world model can track evolutions of the real environment. Building an optimal policy (planning phase) is a quick process, exploiting the current world model.

Indirect reinforcement learning approaches have been introduced by Sutton with the Dyna architecture (Sutton, 1991). The Dyna system is composed of three *asynchronous* tasks:

1. learn a world model
2. build a policy from this world model
3. build a policy from the real world interactions (not using the world model).

The first task is executed in parallel with the two others. The agent is acting and learning either in the real world or in the world model (Dyna switch) as illustrated in figure 4.

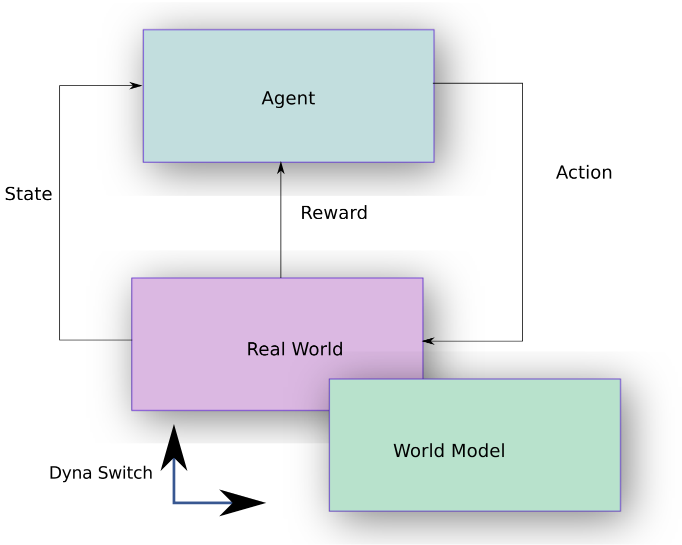


Fig. 4. The Dyna architecture: the agent is learning either on the real world or the world model

5. Our approach

5.1 Indirect Reinforcement Learning adaptation

As stated in section 3.3, learning the user preferences must not be a burden on the end-user. The system has to learn quickly and without over-soliciting the user. Indirect reinforcement learning, presented section 4.4, proposes a solution by using a world models. Applying Dyna implies describing two parts: interactions with the user in the real environment and the non-interactive learning part using world models.

In the interactive part, the system perceives events from the environment through its sensors. The internal representation of the environment's state is updated accordingly (see section 5.2 for further details on state representation). As a reaction to the state change (caused by the detected event), the system selects an action to be executed through its actuators.

The non-interactive part consists in running episodes of Q-Learning in the world model (composed of a transition function and a reward function). Instead of sending actions to the real environment, we query the transition function for the next state given the last action and state. Similarly, the reward function returns us the expected reward for action a in state s. Those functions have to be representative of the real environment's dynamics, which are not fixed but evolving over time. Acquiring them through supervised learning seems appropriate. This learning will be a life-long process in order to keep the models up-to-date with the environment. The learning of the transition function is described section 5.3.1 and the learning of the reward function is the concern of section 5.3.2.

5.2 Internal state representation

One of our constraints defined section 3.3 concerns the intelligibility of the agent (constraint a). The internal functioning of the system should be transparent to the user for him to trust the agent. The modelling choice of environment states should take this constraint into account. We chose to represent a state as a set of *predicates*. Each predicate represents a relevant part of the environment. Zero-order predicates would not provide a sufficiently informative description of the environment because of its complexity. We use first-order predicates, defined with arguments which can take any value or no value. A state is a particular assignment of argument values, which may be null. These predicates are described below.

alarm(title, hour, minute) A reminder fired by the user's agenda.
xActivity(machine, isActive) The activity of the X server of a machine.
inOffice(user, office) Indicates the office that a user is in, if known, null otherwise.
absent(user) States that a user is currently absent from his office.
hasUnreadMail(from, to, subject, body) The latest new email received by the user.
entrance(isAlone, friendlyName, btAddress) Expresses that a bluetooth device just entered the user's office. *isAlone* tells if the user was alone or not before the event.
exit(isAlone, friendlyName, btAddress) Someone just left the user's office.
task(taskName) The task that the user is currently working on.
user(login), userOffice(office, login), userMachine(machine, login) The main user of the assistant, his office and main personal computer (not meant to be modified).
computerState(machine, isScreenLocked, isMusicPaused) Describes the state of the user's computer regarding the screen saver and the music.

An example would be the state:

```
alarm(minute=<null>, title=<null>, hour=<null>);
xActivity(isActive=<null>, machine=<null>);
inOffice(office=<null>, user=<null>);
absent(user=<null>);
hasUnreadMail(from=<null>, to=<null>, body=<null>,
      subject=<null>);
entrance(isAlone=<null>, friendlyName=<null>,
      btAddress=<null>);
exit(isAlone=false, friendlyName=Sonia,
      btAddress=00:12:47:C9:F2:AC);
task(taskName=<null>);
screenLocked(machine=<null>, isLocked=<null>);
musicPaused(isPaused=<null>, machine=<null>);
user(login=zaidenbe);
userOffice(office=E214, login=zaidenbe);
userMachine(login=zaidenbe, machine=hyperion);
```

In this example, the main user is `zaidenbe` and a bluetooth device just left the office.
Each predicate is endowed with a timestamp accounting for the number of steps since the last value change. Among other things, this is used to maintain integrity of states, *e.g.* the predicate `alarm` can keep a value only for one step and only one of `inOffice` and `absent` can have non-null values.
Our states contain free values. Therefore, our state space is very large. This exact information is not always relevant for choosing an action. The user might wish for the music to stop

when anyone enters the office, but to be informed of emails only from his boss. As soon as we observe the state "Bob entered the office", we have an estimated behavior for the state "someone entered the office", which is more satisfying for the user. We generalize states in the behavior definition by replacing values with wildcards: "<+>" means any value but "<null>" and "<∗>" means any value.

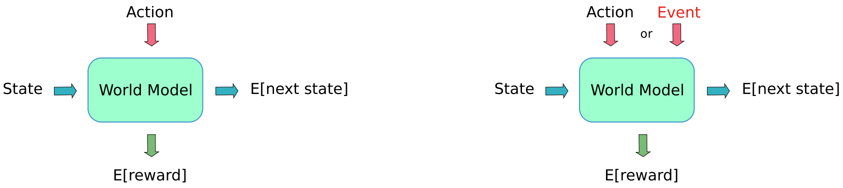
In this manner, an action is associated to a "super-state" encompassing numerous actual states. A generalized state may be split when it is relevant to distinguish between encompassed, more particular, states. This splitting can be done offline, by analyzing the history of rewards and detecting persistent inconsistencies of user rewards for a state. This aspect has not been further studied yet.

5.3 World model

The world model is intended to replace the real world in part of the actions executed for exploration, as shown figure 4. In classical reinforcement learning, the world model takes as input an action executed by the agent and the state that the world was in at the time the action was chosen. The output of the model is the state of the environment after the action and the expected reward (see section 4.4).

In our case, as explained section 4.3.2, the environment state is modified by actions from the agent as well as by exterior events generated by the user. Our world model takes as input the current state and an action *or* an event.

The world model is composed of the transition and reward functions $\mathcal{P}$ and $\mathcal{R}$. We modify slightly the classical definition given section 4.1 by the following definition of the transition function: $\mathcal{P} : \mathcal{S} \times \mathcal{O} \times \mathcal{S} \to [0;1]$, where $\mathcal{O} = \mathcal{A} \cup \mathcal{E}$ represents the set of occurrences, an occurrence being an action or an event ($\mathcal{E}$ is the set of events), as illustrated figures 5(a) and 5(b).



(a) The classical world model (b) Our world model

Fig. 5. World model definition

The world model is automatically acquired using supervised learning based on real interactions examples. The system *records every state, event and reward and action*, and uses these examples for supervised learning as described in sections 5.3.1 and 5.3.2.

5.3.1 Supervised learning of the transition function

In our case, the transition function $\mathcal{P}(s,o,s')$ gives the probability of being in situation s' when applying action a, or perceiving event e (where o is an occurrence a or e), in situation s. We modeled our transition function as a set of transformations. Each transformation includes a starting state s^t, an occurrence o^t, a probability p^t and the modifications that will be applied on the observed starting state to compute the next state $M^t = \{m_i^t, i \in [1;n]\}$.

A modification operates on an predicate's argument. It can erase the argument's value, set a given value, copy the value of one of predicate's `a` argument into one of predicate's `b` argument. The transformation can also be to reset the timestamp. We denote a transformation by $t(s^t, o^t, p^t, M^t)$.

In the same way as we factorize states for the behavior definition (see section 5.2), we use generic transformations in order to reduce the size of the transition function. The starting state s^t is a generalized state, defined with wildcards (for instance, "some entered the office", no matter what the other argument values are).

The transition function is initialized using common sense, and it is enriched by the supervised learning algorithm 1. For instance, consider: $s^t = <\star>$, which matches any state, $a^t =$ `lockScreen`, the action of activating the screen saver. The transformation would be to set to `true` argument `isScreenLocked` of predicate `computerState`, indicating that the screen is locked in the next state.

Algorithm 1: The supervised learning of the transition function.

```
Input: A set of examples $\{s, o, s'\}$
Output: $\mathcal{P}$
foreach example $\{s, o, s'\}$ do
    if a transformation $t$ that obtains $s'$ from $s$ with the occurrence $o$, can be found then
        Increase the probability of $t$;
    else
        Create a transformation starting with $s$, having the action $a$ or a generic event created from $e$ and ending in $s'$, with a low probability;
        Decrease the probability of any other transformation $t'$ that matches the starting state $s$ and the occurrence $o$ but whose ending state is different from $s'$;
```

When new examples confirm knowledge already included in the model, they strengthen the corresponding transformation. If an example contradicts the existing model, we decrease slowly the probability of the concerned transformation. As ac consequence, if this example is an outlier, it will not disturb the model too much. If it is a sign of an actual change, then it will be observed again and will slowly change the model.

This model has a generalization ability since we create new transformations with generic starting states. This accelerated the learning of preferences because we make most use of each example. Having seen one example with the starting state "Bob entered the office", we have a transformation for all states where someone enters the office. This transformation can efficiently be used during offline reinforcement learning.

5.3.2 Supervised learning of the reward function

The reward function $\mathcal{R}(s,a,s')$ indicates the opportunity of choosing action a in situation s. Similarly to the model for the transition function, the model for the reward function is a set of entries, each entry being defined by a starting state s^e, an action a^e and a numerical reward r^e. The given reward usually depends only on a subset of the predicates defining a state. For example if in the starting state the user was typing on his computer and the action is to lock the screen, the reward should be quite negative and does not depend on the presence of other people or on the state of the music player. The starting state s^e may be a generalized state, indicating only relevant filtering constraints. Though, during learning we cannot determine automatically which predicate is relevant for a given reward, since the reward depends on the internal state of the user. The reward model, as well as the transition model, is initialized using common sense. Only these initial entries have generalized starting states. New entries are created with the exact perceived state, without generalization. The model is then learned with the supervised learning algorithm 2.

Algorithm 2: The supervised learning of the reward function.

```
Input: A set of examples $\{s, a, r\}$
Output: $\mathcal{R}$
foreach example $\{s, a, r\}$ do
    if an entry $e = \{s_e, a_e, r_e\}$ such as $s$ matches $s_e$ and $a = a_e$, can be found then
        Update $e$, set $r_e = mix(r, r_e)$, where $mix$ is a merging function;
    else
        Add a new entry $e = \{s, a, r\}$ to the reward model;
```

The merging function used in algorithm 2 amalgamates a newly received reward with the one present in the model for the entry corresponding to the current example. As for the transition function, we want the reward model to evolve smoothly and to be stable. We define $mix(r, r_e) = 0.3r + 0.7r_e$. If the user changes his reward because he changes his preferences, the model will slowly follow the change. If a given reward is inconsistent for another reason, the model will not undergo an unwanted change.

Having numerous user rewards makes the reward model more reliable, but users do not always give rewards. To increase the number of examples, it is possible to infer indirect rewards. For instance, if the user immediately refuses a service proposed by the assistant, for example by closing the mail client opened automatically, we deduce a negative reward. The magnitude of values for indirect rewards is limited to take into account their uncertainty.

5.4 Offline Reinforcement Learning

The behavior learning is done completely offline, in a non interactive manner. We use a classical algorithm for reinforcement learning: the Q-Learning algorithm (Watkins & Dayan, 1992), and we modify it slightly to match our pattern "event-action" (see algorithm 3).

Algorithm 3: An episode of Q-Learning run during the planing stage by the assistant.

```
Input: $\mathcal{P}, \mathcal{R}$
Select an initial state $s$;
for $i$ from $1$ to $k$ do
    Choose an event $e$;
    Send the current state $s$ and the event $e$ to the world model and receive an estimation of the next state $s'$:
    $s' \leftarrow max_{s' \in \mathcal{S}} \mathcal{P}(s'|s, e)$;
    Select an action $a = \pi(s')$;
    Send $s'$ and $a$ to the world model and receive estimations of the next state $s''$ and the reward $r$:
    $s'' \leftarrow max_{s'' \in \mathcal{S}} \mathcal{P}(s''|s', a), r \leftarrow \mathcal{R}(s', a)$;
    Apply a reinforcement learning method on the hypothetical experience $\langle s', s'', a, r \rangle$:
    $Q(s',a) \leftarrow Q(s',a) + \alpha(r + \gamma \max_{a'} Q(s'',a') - Q(s',a))$;
    $s \leftarrow s''$;
    $i \leftarrow i + 1$;
```

Unlike classical Dyna (section 4.4), we use only the world model for learning the behavior. The user acts in the real world and interaction histories are stored for learning the world model. In this way, we can completely separate the learned behavior from the used behavior at a given point in time. The user, who is confronted with the behavior of the assistant, implicity learns his own internal model of the assistant, as he does with every system he uses. If this behavior is constantly updated, the user might get destabilized and lose trust (see constraint a section 3.3). If the learned behavior is not the same as the one used for interactions, it becomes possible to warn the user before replacing the online behavior by the newly updated one.

In algorithm 3, k is the number of iterations of an episode. Each episode starts in an initial state which can be either an empty state (all arguments are `<null>`) or a randomly generated state, or a state randomly chosen from history. The two latter options enable a better exploration of the state space than the first option. Starting from a random state that has never been observed makes sense only because our transition model has a generalization ability. It can estimate a transition from a state never seen during learning (section 5.3.1). Starting from a state encountered during interaction allows to make most use of observed experience. During episodes, events cause state transitions. Likewise, events can be randomly generated or selected from history. Generating unseen events enables better exploration whereas using observed events strengthens past experience.

The world model is initialized using common sense. Before activating the assistant, we run initial episodes of Q-Learning (algorithm 3) to "convert" the initial world model into an initial behavior, to respect constraint e defined section 3.3.

6. Experimental results

6.1 Learning an initial behavior

As mentioned section 5.4, before starting to act, the assistant learns an initial behavior from a given initial world model. For these initial episodes of Q-Learning (algorithm 3) there are four parameters introduced section 5.4 to fix: the number of episodes, the number of iterations of each episode (k), the choice of the initial state and the choice of events during episodes. We ran initial episodes with different options. First, we found that it was more efficient to select both initial states and events randomly from history. Then, we ran 100 episodes of 10, 25, 50 and 100 iterations each and computed a grade for every behavior obtained after an episode. The results are shown figure 6.

The grade of a behavior is for the most part its correspondence to what the user wants. The experimenter who gave the initial rewards is presented with the best action in each state and indicates whether this actions is the one he expected. In a much smaller proportion, the grade takes into account the number of states where the system has an estimation of the action to select. It allows to keep behaviors that have performed enough states explorations (that is, the selected action will not be random in more states).

The curves figure 6 show that the more episodes we run, and the longer the episodes, the better the behavior. But this figure does not show the computation time needed for each point in the curves. Episodes with 100 iterations take twice as long as episodes with 50 iterations whereas the grade is only approximately 1.1 times higher in the end. Hence, we choose to keep $k = 50$ iterations per episode. The same principle applies to the number of episodes. During the first 50 episodes, the grade raises notably. The last 50 episodes take, again, twice as long and result in a grade only 1.2 times higher. Therefore, the assistant can start acting after 50 episodes of 50 iterations, its behavior will be acceptably good. Anyway, after this initial learning, episodes are run regularly and the behavior continues enhancing. This result is presented section 6.2.

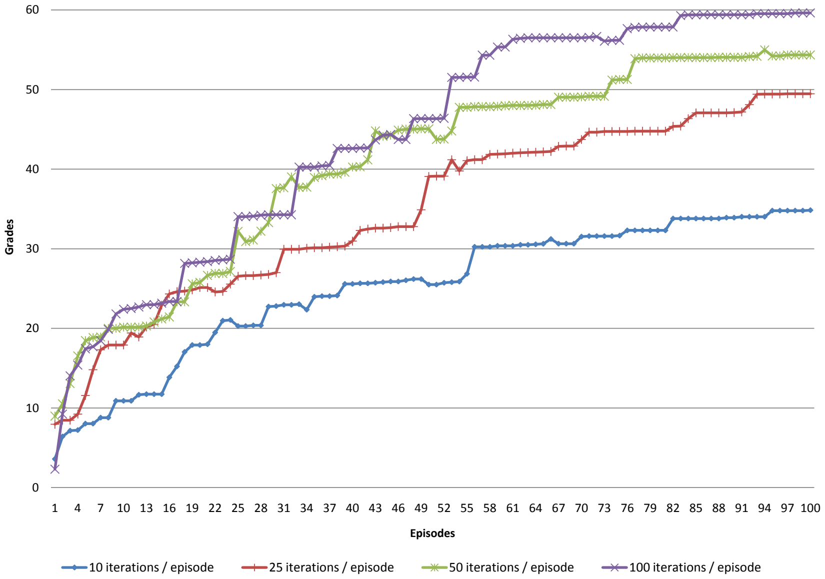


Fig. 6. Initial episodes with events and initial states randomly selected from history.

6.2 Learning user preferences during interactions

Once an initial behavior was acquired, we performed an experience to measure the efficiency of learning during the “normal” life of the assistant with its user. The role of the user was taken by a human experimenter and consisted in “generating” real life events, such as leaving and entering the office, receiving emails and reminders, having other people coming in and out of the office, playing or pausing the music and starting and stopping the screensaver. To send these events to the assistant, the experimenter used a graphical interface, facilitating him the experience. He was also in charge of giving rewards to the assistant. For that, the experimenter had a goal behavior in mind and gave rewards mostly consistent with this goal. Like a normal user, the experimenter did not give rewards for all actions. During these interactions, at regular time intervals, reinforcement learning episodes were executed and the resulting behavior was evaluated by the experimenter. As previously, he was interviewed about the best action in every state and answered whether this action was the one he had in mind or not. The resulting grades are shown figure 7.

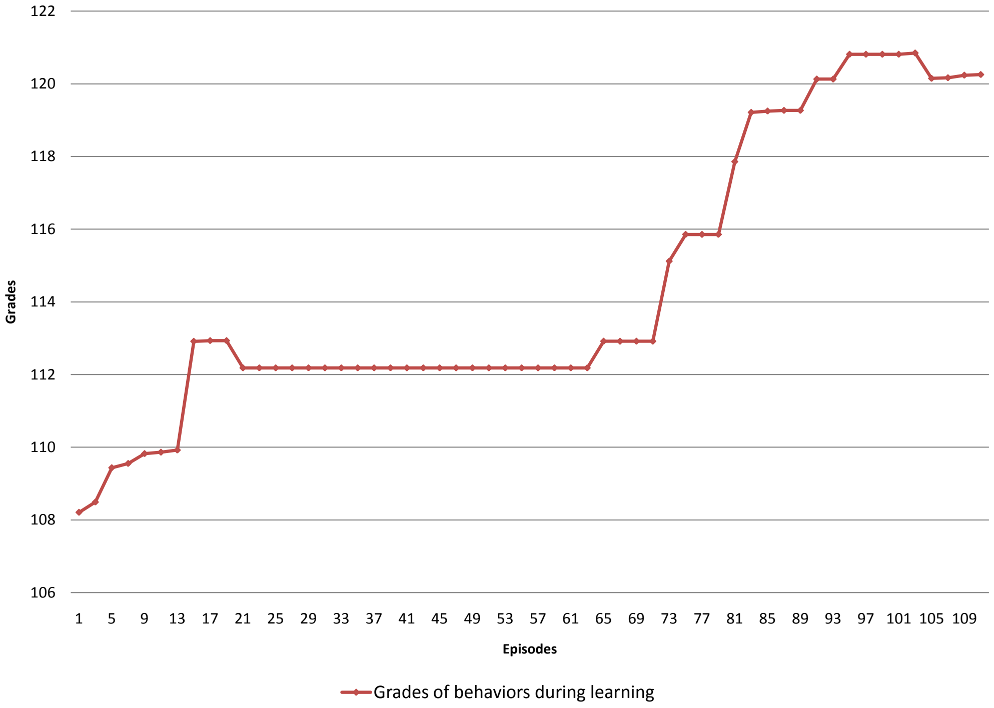


Fig. 7. Grades of behaviors produced by the reinforcement learning algorithm.

The periods in figure 7 where the curve is rising correspond to periods where user events were new to the assistant (they were not included in the initial world model). The curve is flat when the learned behavior is optimal for the time being.
Figure 8 shows an example where the experimenter changed his mind on a part of the expected behavior. The grade drops and then the system adapts to the new reinforcements.

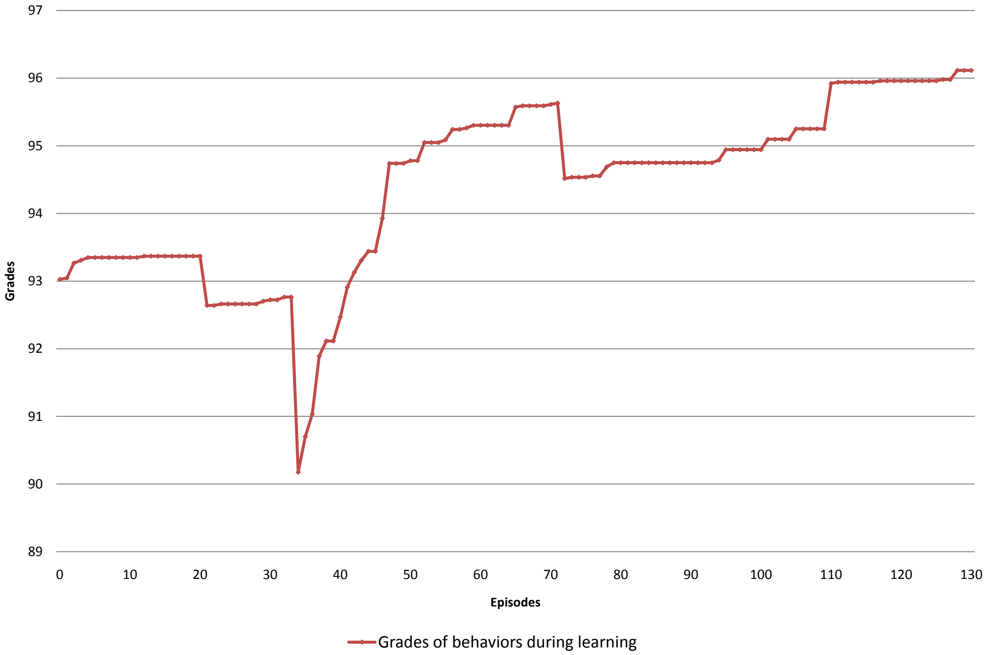


Fig. 8. Grades of behaviors produced by the reinforcement learning algorithm. Example of preference change.

7. Conclusion

The aim of this research is to investigate Ambient Intelligence systems and their acceptability by users. We exploit a ubiquitous system to provide personalized, context-aware services to users. The personalization of the system is achieved by learning user preferences during interactions. We proposed a reinforcement learning method corresponding to the pattern "event-action" existing in ubiquitous systems. The classical reinforcement learning or indirect reinforcement learning is not adapted directly to real world applications where the user is in the loop. We conducted a user study in order to validate the relevance of such an application and reveal constraints for the acceptability of such a system. We adapted existing methods to those requirements and developed a prototype showing the correct functioning of our ubiquitous assistant.

8. References

Abowd, G. D., Atkeson, C. G., Feinstein, A., Hmelo, C., Kooper, R., Long, S., Sawhney, N. & Tani, M. (1996). Teaching and learning as multimedia authoring: the classroom 2000 project, *Proceedings of the fourth ACM international conference on Multimedia*, ACM, Boston, Massachusetts, United States, pp. 187–198.
URL: *http://portal.acm.org/citation.cfm?doid=244130.244191*

Abowd, G. D. & Mynatt, E. D. (2000). Charting past, present, and future research in ubiquitous computing, *ACM Trans. Comput.-Hum. Interact.* **7**(1): 29–58.
URL: *http://portal.acm.org/citation.cfm?id=344988*

Adams, N., Gold, R., Schilit, B. N., Tso, M. & Want, R. (1993). An infrared network for mobile computers, *Proceedings USENIX Symposium on Mobile & Location-independent Computing*, p. 41–52.

Aström, K. J. (1965). Optimal control of markov decision processes with incomplete state estimation, *Journal of Mathematical Analysis and Applications* **10**: 174–205.

Bellotti, V. & Edwards, K. (2001). Intelligibility and accountability: human considerations in context-aware systems, *Hum.-Comput. Interact.* **16**(2): 193–212.
URL: *http://portal.acm.org/citation.cfm?id=1463108.1463113*

Brdiczka, O. (2007). *Learning Situation Models for Providing Context-Aware Services*, PhD thesis, Institut National Polytechnique de Grenoble (INPG).
URL: *http://www-prima.inrialpes.fr/prima/pub/Publications/2007/Brd07/*

Brumitt, B., Meyers, B., Krumm, J., Kern, A. & Shafer, S. (2000). EasyLiving: technologies for intelligent environments, *Handheld and Ubiquitous Computing*, pp. 97–119.
URL: *http://dx.doi.org/10.1007/3-540-39959-3_2*

Buffet, O. (2003). *Une double approche modulaire de l'apprentissage par renforcement pour des agents intelligents adaptatifs*, PhD thesis, Université Henri Poincaré, Nancy 1. Laboratoire Lorrain de recherche en informatique et ses applications (LORIA).

Byun, H. E. & Cheverst, K. (2001). Exploiting user models and Context-Awareness to support personal daily activities, *PERSONAL DAILY ACTIVITIES, WORKSHOP IN UM2001 ON USER MODELLING FOR CONTEXT-AWARE APPLICATIONS* pp. 13—16.
URL: *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.4677*

Cheverst, K., Byun, H., Fitton, D., Sas, C., Kray, C. & Villar, N. (2005). Exploring issues of user model transparency and proactive behaviour in an office environment control system, *User Modeling and User-Adapted Interaction* **15**(3): 235–273.
URL: *http://dx.doi.org/10.1007/s11257-005-1269-8*

Coen, M. H. (1998). Design principles for intelligent environments, *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, American Association for Artificial Intelligence, Madison, Wisconsin, United States, pp. 547–554.
URL: *http://portal.acm.org/citation.cfm?id=295733*

Craik, K. (1943). *The Nature of Exploration*, Cambridge University Press, Cambridge, England.

Crowley, J., Coutaz, J., Rey, G. & Reignier, P. (2002). Perceptual components for context aware computing, *UbiComp 2002: Ubiquitous Computing*, pp. 117–134.
URL: *http://dx.doi.org/10.1007/3-540-45809-3_9*

Dey, A. K., Hamid, R., Beckmann, C., Li, I. & Hsu, D. (2004). a CAPpella: programming by demonstration of context-aware applications, ACM, Vienna, Austria, pp. 33–40.
URL: *http://portal.acm.org/citation.cfm?id=985692.985697*

Intille, S. S. (2002). Designing a home of the future, *IEEE Pervasive Computing* **1**(2): 76–82.

Kaelbling, L. P. (2004). Life-Sized learning. Published: Lecture at CSE Colloquia.
URL: *http://www.uwtv.org/programs/displayevent.aspx?rID=3566*

Kay, J., Kummerfeld, B. & Lauder, P. (2003). Managing private user models and shared personas, *UM03 Workshop on User Modeling for Ubiquitous Computing*, p. 1–11.

Kozierok, R. & Maes, P. (1993). A learning interface agent for scheduling meetings, *IUI '93: Proceedings of the 1st international conference on Intelligent user interfaces*, ACM Press, New York, NY, USA, p. 81–88.
URL: *http://portal.acm.org/citation.cfm?id=169908*

Littman, M. L. (1994). Markov games as a framework for Multi-Agent reinforcement learning, *IN PROCEEDINGS OF THE ELEVENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING*, pp. 157—163.
URL: *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.8623*

Lueg, C. (2002). Looking under the rug: On Context-Aware artifacts and socially adept technologies, *Proceedings of Workshop "The Philosophy and Design of Socially Adept Technologies", ACM SIGCHI Conference on Human Factors in Computing Systems*.

Maes, P. (1994). Agents that reduce work and information overload, *Com. ACM* **37**(7): 30–40.
URL: *http://portal.acm.org/citation.cfm?id=176792*

Michaël, M. (1998). The neural network house : An environment that adapts to its inhabitants, *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, AAAI Press, Menlo Park, CA, pp. 110–114.

Muir, B. M. & Moray, N. (1996). Trust in automation. part II. experimental studies of trust and human intervention in a process control simulation, *Ergonomics* **39**(3): 429.
URL: *http://www.informaworld.com/10.1080/00140139608964474*

Papadimitriou, C. H. & Tsitsiklis, J. N. (1987). The complexity of markov decision processes, *Mathematics of Operations Research* **12**(3): 441–450.
URL: *http://www.mit.edu/ jnt/publ.html*

Pineau, J., Gordon, G. & Thrun, S. (2003). Point-based value iteration: An anytime algorithm for pomdps, *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1025–1032.
URL: *http://www-old.ri.cmu.edu/pubs/pub_4826.html#text_ref*

POMDPs for Dummies: Page 5 (1999).
URL: *http://www.cs.brown.edu/research/ai/pomdp/tutorial/pomdp-solving.html*

Remagnino, P. & Foresti, G. (2005). Ambient intelligence: A new multidisciplinary paradigm, *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* **35**(1): 1–6.

Rogers, Y. (2006). Moving on from weiser's vision of calm computing: Engaging UbiComp experiences, *UbiComp 2006: Ubiquitous Computing*, pp. 404–421.
URL: *http://dx.doi.org/10.1007/11853565_24*

Rummery, G. A. & Niranjan, M. (1994). On-line q-learning using connectionist systems, *Technical report*, Cambridge University Engineering Department.

Salovaara, A. & Oulasvirta, A. (2004). Six modes of proactive resource management: a user-centric typology for proactive behaviors, *Proceedings of the third Nordic conference on Human-computer interaction*, ACM, Tampere, Finland, pp. 57–60.
URL: *http://portal.acm.org/citation.cfm?id=1028014.1028022*

Smith, T. & Simmons, R. G. (2005). Point-based POMDP algorithms: Improved analysis and implementation, *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*.
URL: *http://www.cs.cmu.edu/ trey/pubs/b2hd-smith05hsvi.html*

Spaan, M. T. J. & Vlassis, N. (2005). Perseus: Randomized point-based value iteration for pomdps, *Journal of Artificial Intelligence Research* **24**: 195–220.
URL: *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.72.9217*

Sutton, R. (1988). Learning to Predict by the Methods of Temporal Differences, *Machine Learning* **3**: 9–44.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting, *SIGART Bull.* **2**(4): 160–163.
URL: *http://portal.acm.org/citation.cfm?id=122377*

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*, The MIT Press.

Walker, M. A. (2000). An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email.
URL: *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.1121*

Watkins, C. & Dayan, P. (1992). Q-Learning, **8**(3-4): 279–292. ISSN 0885-6125.

Weiser, M. (1991). The computer for the 21st century, *Scientific American* .
URL: *http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html*