Mixed Initiative Ambient Environments: A Self-Learning System to Support User Tasks in Interactive Environments

Kristof Verpoorten*, Kris Luyten, and Karin Coninx

Hasselt University
Expertise Centre for Digital Media
and transnationale Universiteit Limburg
Wetenschapspark 2
BE-3590 Diepenbeek, Belgium
{kristof.verpoorten,kris.luyten,karin.coninx}@uhasselt.be

Abstract. In this paper we introduce a pro-active self-learning agent system that supports the user in an ambient intelligence environment. Interaction in an ambient intelligence environment is typically very complex and demands lots of attention of the end-user. In many cases it is extremely difficult for the user to evaluate what the consequences are from interactions accomplished in this space. In the system we present in this paper, an agent monitors the user's interactions with the environment to learn the user's expectations when certain activities are performed. Machine learning techniques allow the system to pro-actively react on behalf of the user. When the system is not entirely sure whether the user wants to execute a certain action, it will not execute it on his or her behalf, but makes it easier for the user to execute it himself by guiding the user through the options. This mixed initiative can help the user while not risking to lose his or her confidence by executing the wrong action on the user's behalf. The kind of system presented in this paper takes a lot of simple, often reoccurring work out of the user's hands.

1 Introduction

A lot of our daily actions, are reactions to things happening around us. Often, these action-reaction pairs are quite simple and do not require a lot of background knowledge to execute properly. Because of this, a computer agent should be able to "learn" the proper response to certain actions by monitoring interactions between your device, your environment and yourself.

One can imagine several scenarios in which this kind of system would be useful. When the user arrives at an unknown environment, he or she does not know what the possibilities of that environment are. Our agent will also help to overcome this uncertainty of the user. When the agent detects that the user wants to execute certain tasks with a high probability in the ambient intelligence environment, it will adapt the user interface to guide the user through the required interactions to accomplish this task

* Author supported by the Fund For Scientific Research Flanders (F.W.O. Vlaanderen).

based on previously learned data. Another scenario we had in mind when developing the system is the following. Imagine a manager of a large company using a smartphone to schedule meetings, send e-mails, make phone calls, show presentations etc. This smartphone is essentially a PDA with phone support. It supports several wireless communications protocols like WiFi, Bluetooth, GSM, GPRS, ... An example of such a smartphone is the Qtek 9100. This manager carries the device with him at all times, when travelling, in his office or at meetings. He also uses it to give presentations in meetings he needs to attend in his corporate building. For this purpose, he has to start the slide show software on the phone and connect it with the projector in the meeting room using Bluetooth. Of course it is also important that the volume of the PDA is turned down, in order to avoid interruptions by incoming phone calls during the meeting. When leaving the meeting room he also has to remember to turn on the sound of the phone again.

This is an ideal task for a pro-active agent to take over from the user. In the meeting room there is a router for wireless internet access and a Bluetooth projector used for presentations. Both devices can be detected by our system using WiFi or Bluetooth. Now imagine the next situation: the manager enters the meeting room. The first thing he does is to turn the volume of his PDA down. Next, he gets to the front of the room and opens the presentation on his PDA. He also establishes a Bluetooth connection with the projector. After the meeting he leaves the meeting room and turns the audio volume of his PDA up again. The next time he enters this same meeting room, his personal device immediately recognizes the access point, and remembers that last time it used this router, its sound capability was disabled. Because this only happened once before, the device does not turn off the sound automatically, but adapts the user interface to make it easier for the manager to turn off the sound (e.g. show a button that mutes the sound with one click) and asks the manager whether or not that is what he wants to do. When the manager confirms this, the sound is effectively turned off. This also happens when he opens a presentation on his PDA. The agent on the device has already detected the Bluetooth projector and remembers that last time, the manager established a Bluetooth connection with the projector after he opened the presentation. Again, the device asks if the user wants it to make this connection and makes it easier for him to do so by adapting the user interface. When leaving the room after the meeting, the device will also notice the router going out of range, and the agent will suggest to turn on the sound again. The third time the manager enters the meeting room, the device will again recognize these previously encountered situations and make suggestions or even take control into its own hands when it is very sure about the right action to take (turn off the sound when entering the meeting room without asking first, or connecting with the projector automatically when the presentation is opened).

This scenario shows the use of the system presented in this paper. The system will continuously learn which user actions can be linked with certain resources (e.g. other devices or available network connections) being in the vicinity or not. Next time, when the same situation presents itself, the system will be able to execute simple actions on behalf of the user or guide the user in interacting with the environment to accomplish her or his goals.

Our system learns from the user by monitoring his or her actions and monitoring the *context* of the user. Anind K. Dey defines context as follows [Dey00]:

> "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves."
>
> — Anind K. Dey

We consider as context all information that can be detected by the user's personal device and that can be relevant when trying to predict the user's next action(s).

The remainder of this paper is structured as follows: section 2 will elaborate on some related work. Next, section 3 will explain how our system represents and manages the ever changing environment around the user. Using this representation of the environment, the agent is able to learn from it, and make predictions regarding the user's next action(s). The operation of this part of the system is explained in section 4. Our system cannot always be certain about the action(s) the user is going to do next, but in that case, it can still adapt the user interface (UI) to help the user execute his or her actions. In section 5 you can read how adaptation of the UI can also help the user execute actions he or she did not even knew were possible in the environment. Finally, section 6 will outline the conclusions.

2 Related Work

Pattie Maes [Mae95] suggests the idea of agents acting as personal assistants. These agents acquire their competence by learning from the user, as well as learning from agents assisting other users. A few prototypes were build using this technique, including agents that provide assistance with meeting scheduling, e-mail handling, news filtering and entertainment selection. The problem with this approach is that the agents will only monitor one program (agenda, e-mail client, ...). We think more usefull results can be obtained by monitoring the entire environment and the user's personal device.

Charles Isbell et al [CLIOP04] try to predict which task the user wants to execute with a remote control. They deploy a user interface that allows the user to execute a task using only one remote to control several devices. They describe tasks as clusters of similar commands, which are often used together. With the recorded data of 2 users during several weeks they were able to collect enough interactions to divide all available commands into several clusters. Each cluster represents a task the user is performing, during which he or she uses the commands in the cluster. They predict the next task the user is likely to perform by looking at the previous tasks he or she has performed. That way the user interface of the remote is adapted to support the probable next task, avoiding a cluttered UI with to many buttons. The way they predict the next task has some similarities with our approach of sliding windows (section 4.3). Another resemblance with our approach is that when our agent is not entirely sure about the next action to take, it will not execute an action on behalf of the user, but adapt the UI on the user's

device to make it easier for the user to execute the task himself if desired. However, in contrast to us, they only focus on devices that can be controlled using a remote. Our system is designed to support the user with almost everything he or she can do using his or her PDA. Our system will also take over easy tasks from the user when it has had enough training, while the remote developed by Isbell et al will only adapt the UI of the remote to help the user in executing his or her task.

Pro-active interfaces can already be found in several prototype applications, as demonstrated by Lieberman et al. As consumer electronics are becoming more complicated and a lot of users are unable to understand their full potential, they created 'Roadie' [LE06]; a prototype interface that will try to infer the user's goal by monitoring his or her actions. To make predictions, it will use EventNet [EL05], a plan recognizer that uses knowledge mined from the 'OpenMind Commonsense knowledge Base' [SLM+02], a knowledge base describing everyday life. Using Roadie, users can select their goal from a list of suggestions. Next, planning and commonsense reasoning is used to explain how to reach the goal. The answer is displayed in plain text, explaining the user's next action(s). If the user makes a mistake, Roadie will launch a debugging dialog for extra assistance.

Alternatively, Dey et al [DHB+04] propose *a CAPpella*, which uses programming by demonstration to teach a context-aware application. This program captures raw context data, which can be replayed afterwards to indicate the relevant parts. After several iterations of recording and indicating the relevant parts, the application should be able to recognize a certain context (e.g. a meeting), and hence perform the relevant actions for that context (e.g. launching a notepad to make notes during a meeting). The value of this work is that it learns to recognize a context from natural data; however the user still has to indicate which are the relevant parts. This is a burden on the user that we try to overcome in our system. By using decision trees, irrelevant context data will automatically be filtered out when similar situations have happened where only the relevant data and the outcome were the same.

Byun and Cheverst propose the utilization of context history together with user modeling and machine learning techniques to create pro-active applications. In [BC04] they describe an experiment to examine the feasibility of their approach for supporting pro-active adaptations in the context of an intelligent office environment. The application uses two different approaches to obtain pro-activity: *pro-active rule based adaptation*, and *pro-active modeling adaptation*. With the first approach, users have to reconfigure the system as their preferences change. The second approach automatically adapts those predefined rules when observation makes clear that the user's preferences have changed. In order to learn the patterns of the user's behavior, *decision trees* are used. Decision trees have the advantage to be much easier to understand by designers than other machine learning techniques such as neural networks. Byun's conclusion is that context history has a concrete role for supporting pro-active adaptation in ubiquitous computing environments. This work supports our decision to use context history as our main source of data when trying to predict the user's next action. Both our decision tree and sliding window techniques (section 4) use context history.

Obviously, a pro-active agent takes some control away from the user. Is the user willing to make this sacrifice? Barkhuus and Dey have examined this very question

in [BD03]. Their conclusion is that the user is willing to accept a large degree of autonomy from applications as long as the application's usefulness is greater than the cost of limited control.

3 Environment

3.1 Representation

For the agent to be able to react to certain events that occur in the environment of the user, it has to be able to monitor this environment. Therefore our system represents the environment of the user in such a way that it is understandable for a computer program. The system monitors the environment continuously (through a set of sensors) and stores all relevant information in a graph-like structure. An example of such a graph is shown in figure 1.

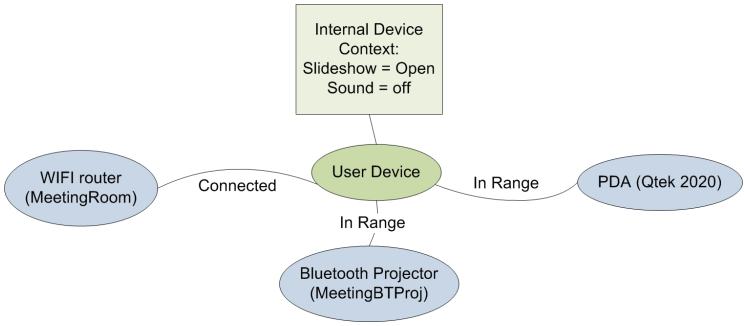

Fig. 1. The environment is stored in a graph-like structure. In the center is the user's device with a link to its internal context. The other edges describe the relation of the user's device with other devices in the vicinity (e.g. in range, connected, ...). This is what the environment could look like for our agent in the scenario explained in the introduction.

Each node of the graph represents a resource (device) in the environment or a property of a resource and the edges represent relations between the resources. The node representing the user's device will be in the center of the graph. It will have several outgoing edges to all other devices in the vicinity that can be of any relevance to the user's context. It will also have an outgoing edge to a node representing the device's internal context. The internal context node contains all relevant information about the device itself (running programs, battery level, ...). The representation of the environment is sufficient for our current needs, but it is still work in progress and will be further extended when necessary (e.g. by using the CoDAMoS ontology [PdBW+04]).

3.2 Management

Often, the environment as perceived by the agent will be dynamic: objects can move around, devices become available or move out of range, other users may enter the same

environment,.... Especially when the agent is running on a portable device that will be changing location from time to time. Therefore the system will continuously monitor the environment in order to detect relevant changes. These changes will then be reflected in the graph that represents the environment. The graph is updated according to the changing context of the user by means of several elementary operations. Each operation allows us to change one elementary part of the graph. These elementary operations are to add/remove a vertex or an edge. Because we use elementary operations it is possible for the agent to keep a context history. In case the user does not agree with an action the agent has taken, the agent can check how the environment looked in the past by executing the list of elementary operations in reversed order, and undo the taken action.

4 Learning and Prediction

4.1 General Overview

To predict the next action the user wants to execute, the agent must first learn when and in which context the user performs these actions. There are often events that occur in the environment and they trigger a user's action. It is those events we want to detect to predict and anticipate possible user reactions. For that purpose the environment is continuously being monitored by the system.

When the user performs an action, the agent can learn that this action occurs in a certain context. This context is described as the environment graph mentioned in section 3. If an action is frequently executed in a certain context, the selection of the specific contexts in which this action can occur becomes more precise. In practice this is accomplished by selecting the relevant subgraphs of the environment graph. This way the agent learns the same action should be executed in other but similar contexts. Since our context for executing an action is defined by a set of graphs, similarity is simply the fact these graphs are included as subgraphs in the current environment graph. In section 4.2 we show a performant way to remove unwanted context information and decide upon the most probable "next action" given a certain context-of-use.

Another way to predict the user's actions, is by looking at the sequence of actions that often occur before this particular action. There are a lot of actions that almost always follow each other or a sequence of other actions. When the system can detect these sequences, it can be reasonably sure something will happen when all previous actions of the sequence have occurred.

These two techniques will be explained in detail in section 4.2 and 4.3. Our system will use both approaches at the same time. Each time the user executes an action, both the decision tree and the sliding window are updated. Next, when the agent will try to predict what action the user is likely to execute next, it will compare the outcomes of both algorithms. If both predictions are the same, the agent can be pretty sure it is correct, if they are different, it will try to adapt the user interface to support one or both of the predicted actions and help the user that way.

4.2 Decision Tree

To train the system, decision trees are used [Mit97]. There are several reasons to prefer decision trees over other machine learning methods (e.g. neural networks or bayesian

learning). One important reason is human readability. Most humans can understand a decision tree by looking at it, which is most certainly not the case with neural networks. Another important reason to prefer decision trees is that they require less training then neural networks.

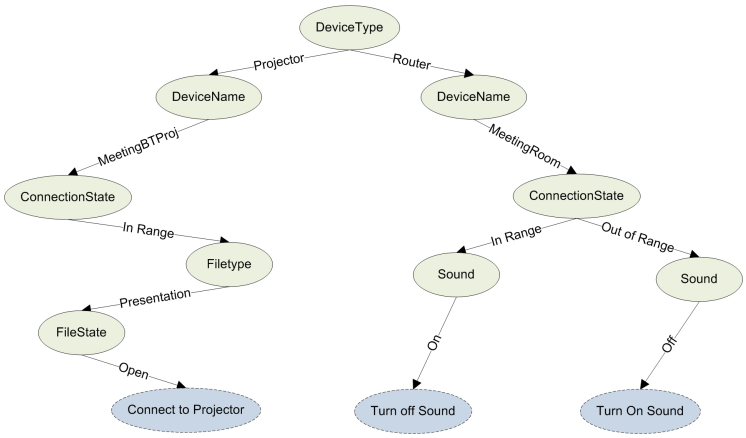

Fig. 2. A trained decision tree capable of supporting the scenario explained in the introduction.

The decision tree is created using information from the environment graph. Each time the user executes an action with the device, the agent will receive an event and use the data from the environment graph to add an entry to the decision tree. Next time a similar context is detected, the decision tree will be used to find the action that will be executed next with the highest probability.

When an action occurs more frequently, the decision tree can be refined by throwing out irrelevant context information. For example: the first time the user executes a certain action, all context information is considered relevant, because the agent cannot know which information is relevant and which is not. Each time the user executes the same action, some of the context information will most likely differ from the previous times. Therefore, the agent knows the different information is not relevant in this situation, and the decision tree can be adapted accordingly. Because after a few times all the irrelevant data will be removed from the decision tree, new entries of the same situation will not trigger any more changes in the tree. When all or most irrelevant information is removed from the decision tree, the system will be able to recognize when a similar situation occurs and react accordingly. This is because only irrelevant information will differ in that case, which is ignored by the system.

4.3 Sliding Window

The second machine learning technique used is a sliding window. With this technique, all actions of the user are encoded as letters in an alphabet and are recorded in a list.

The alphabet contains a symbol for each type of action that can be executed in the environment. After a while, interesting patterns will occur within that list of actions.

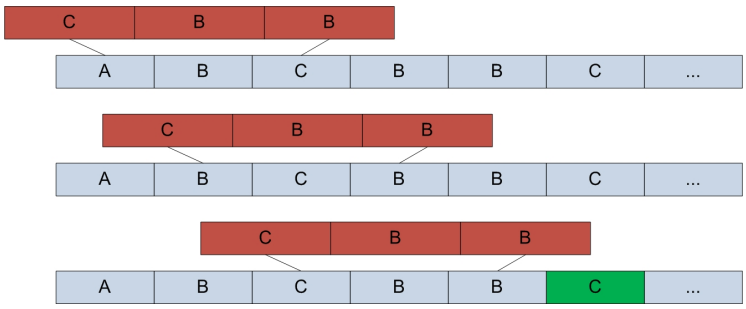

Fig. 3. The window contains the last 3 actions executed by the user. When a match is found in the list of recorded actions, the next action after the match, shown in green, is the prediction that will be used.

When the agent tries to predict the next action a user will execute, it looks for patterns in the recorded list (figure 3). It will do this by creating a window containing the last x actions executed by the user. The system will use several lengths between a maximum and a minimum length for this window. It will first try to find reoccurring patterns using the maximum length for the window. Of course, with a long window, it will be harder to find a match but the likelihood of the predicted action based on this will be higher. When it does not find a match, it will start looking for matches with shorter windows, thus predicting actions with a lower likelihood. This goes on until the system finds a match or it has reached the minimum length for the window. When a matching window is found, the action that occurs directly after the match will probably be the next action the user is going to execute. The reasoning behind this is simple, when the x previous actions are in the same order as the last time we recorded this pattern, probably the $x + 1$'th action will also be the same as the last time.

The length of the window determines the certainty of the prediction. The longer the window, the lower the chance it is a coincidence that the user is executing the same actions in the same order. Imagine there are 5 different recorded actions in the list. Then with a window size of 1, there is 20% chance it is a coincidence the user executed the same action as in the list. With a window size of 2, there is only a 4% chance it is a coincidence. The longer the window, the more sure the agent can be it is not a coincidence and the predicted action will be correct.

5 Mixed Initiative

When the user is working with his or her personal device, our agent is continuously monitoring the environment and the user's actions. After a while, certain situations will be recognized by the agent. When the current situation is recognized by the agent, it has

several options. Depending on how certain the agent is of the user's expectations, it will react differently. When the agent has only observed the current behavior a few times before, it can not be very certain the predicted action is correct. Therefore, it will not execute the action on behalf of the user. If it did, and it would turn out to be undesired behavior, the user would lose confidence in the system. Confidence is very important when working with pro-active agents. The user's confidence can be easily lost when the agent takes bad decisions for the user. Once confidence is lost, it will be very difficult to recover the user's trust in the agent. So it is better to be on the safe side, and only execute actions on behalf of the user when there is a high degree of certainty that the user desires those actions to be executed. Also, the agent has to enable the user to very easily undo the action it has executed (e.g. show an undo button on the screen), so in case the agent takes a wrong decision, the effects of this desision can be easily undone. Although the agent will not execute the action when the degree of certainty does not exceed a threshold (defined experimentally), it can try to help the user by adapting the user interface when this threshold is not reached. The agent can adapt the interface in such a way that it becomes easier for the user to execute the task predicted by the agent. This adaptation can take many forms e.g. highlighting the area of the user interface needed to perform the task, changing focus to that part of the interface, make that part appear larger than other parts of the interface, ... This approach allows the agent to guide the user, while not forcing him or her to execute the predicted action.

This approach can also be of great help when the user is present in an unknown environment. Often, the agent will be able to recognize similarities between this new environment and previous encountered environments. When a few relevant context details happen to be the same in this new environment, the agent can help the user to execute the action she or he would probably want to do, even if the user did not know it was possible. For example: imagine a user that is addicted to e-mail, each opportunity she gets, she will go online to check her e-mail. After a while, the agent will have noticed that each time an unsecured wireless network is in range, the user will download her new e-mails. Now imagine the user has to go abroad for a conference. She is totally unfamiliar with the environment and has no idea where she can find unsecured networks to check her e-mail. Fortunately, the agent has learned enough already and it can guide the user by showing that she can check her e-mail as soon as it finds an unsecured network. If the agent is very sure that the user always wants to do that, it can even download new e-mails on her behalf. This simple example shows how our agent system can help the user find his or her way in unknown ambient intelligence environments by suggesting possible actions the user might want to take, even if he or she did not know it was possible.

6 Conclusions

In this paper we introduced a pro-active, self-learning agent system supporting the user in an ambient intelligence environment. The system continuously monitors the environment and the user's actions with his or her device. Each time the user undertakes an action, the current context data is stored using a decision tree. The action is also added to a list which is used to search for patterns in the user's actions. Using the combination

of pattern matching and decision threes, the agent system predicts which action the user is most likely to execute next. When the probability about the next action the user will take reaches a certain threshold, the agent will execute this action on the user's behalf. The agent stores the history as changes in the context graph so actions done either by the agent or user can be reversed with ease. When the agent is not sure, it will still support the user in his or her task by adapting the user interface so it emphasis the parts of the interface that require interaction to accomplish the task at hand. In ambient intelligence environments where the user interface is spread throughout the environment, this makes the possible ways the user can reach a goal more perceivable.

Acknowledgments

Part of the research at EDM is funded by EFRO (European Fund for Regional Development), the Flemish Government and the Flemish Interdisciplinary institute for Broadband Technology (IBBT). Funding for this research was also provided by the Fund For Scientific Research Flanders (F.W.O. Vlaanderen, project number G.0461.05).

References

[BC04] H.E. Byun and K. Cheverst. Utilising context history to support proactive adaptation. In *Applied Artificial Intelligence, vol. 18, nr. 6*, pages 513–532, July 2004.

[BD03] L. Barkhuus and A. K. Dey. Is context-aware computing taking control away from the user? three levels of interactivity examined. In *Proceedings of UbiComp 2003*, Seattle, Washington, 2003. Springer.

[CLIOP04] Jr. Charles L. Isbell, Olufisayo Omojokun, and Jeffrey S. Pierce. From devices to tasks: automatic task prediction for personalized appliance control. *Personal Ubiquitous Comput.*, 8(3-4):146–153, 2004.

[Dey00] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, November 2000.

[DHB+04] Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. a cappella: Programming by demonstration of context-aware applications. In *Proceedings of CHI 2004*, Vienna, Austria, April 2004.

[EL05] José Espinosa and Henry Lieberman. EventNet: Inferring Temporal Relations Between Commonsense Events. In *Proceedings of MICAI 2005*, pages 61–69, 2005.

[LE06] Henry Lieberman and José Espinosa. A goal-oriented interface to consumer electronics using planning and commonsense reasoning. In *Proceedings of IUI '06*, pages 226–233, New York, NY, USA, 2006. ACM Press.

[Mae95] Pattie Maes. Agents that reduce work and information overload. *Human-computer interaction: toward the year 2000*, pages 811–821, 1995.

[Mit97] Tom Mitchell. *Machine Learning*. McGraw-Hill Education (ISE), October 1997.

[PdBW+04] D. Preuveneers, J. Van den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. De Bosschere. Towards an extensible context ontology for ambient intelligence. In *European Symposium on Ambient Intelligence*, pages 148–159, November 2004.

[SLM+02] Push Singh, Thomas Lin, Erik Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. Open mind common sense: Knowledge acquisition from the general public. In *roceedings of the First International Conference on Ontologies, Databases, and Applications of Semantics for Large Scale Information Systems*, P. Irvine, CA, 2002.