A Context-Aware Proactive Controller for Smart Environments

Frank Krüger* Gernot Ruscher Sebastian Bader Thomas Kirste

Universität Rostock,
Albert-Einstein-Str. 21,
18059 Rostock, Germany
*corresponding author: frank.krueger2@uni-rostock.de

ABSTRACT

In this paper we describe an *implicit* user interface for smart environment control: We make our system *guess* how to assist the user(s) proactively. Our controller is based on two formal descriptions: One that describes user activities, and another that specifies the devices in the environment. Putting both together, we can synthesize a probabilistic model, the states of which resemble activities performed by the user(s) and are annotated with sequences of device actions, with the latter to be executed in cases particular activities have been recognized. The resulting system is purely reactive and can be executed in real time.

Categories and Subject Descriptors

H.5 [**User Interfaces**]: Input Devices and Strategies

General Terms

Theory

Keywords

intention recognition, HMM, planning, smart environments

1. INTRODUCTION

As computers become smaller and smaller, the vision of ubiquitous computing becomes true. At the same time, smart environments contain a large number of devices and become thus more and more complex. Thus, configuration as well as correct usage gets more time consuming and error prone. Exploring new ways to control these *invisible* devices is a challenge addressed by current research [2]. Our approach is to create an entirely reactive system to control all devices of the environment by inferring the intentions of the user. The system gives support by controlling the devices the way the user would do to achieve his *goals*. We use a semantic modeling of the user and the environment to assure that the support is sound and complete, in the sense that the environment is able to support the user correctly in every recognizable situation.

To proactively support users in instrumented environments, we need to infer their intentions, the goals behind their current activities. Here we use a rather technical notion of intention: given descriptions of complex actions, like giving a presentation or preparing a meal. If we detect the user performing some sub-tasks of these complex action, we assume that his goal is to perform the complex action completely. A controller such as described requires all calculation to be executed in realtime. It is therefore necessary to move time consuming operations like planning processes from runtime to compile time. Thus, we can create a purely reactive controller with time-bounded complexity, able to control the environment in every possible situation.

As illustrating example in this paper we use the task of giving a presentation inside our smart meeting room. This environment is introduced below. The graphical representation of this task is given in Figure 1. Here the task of giving a presentation decomposes to a sequence of sub-tasks. The user starts the presentation with entering the room and moving to the front of the room. When the presentation is finished the user moves to the door to leave the room. A more detailed description of this example is given in section 3.

2. PRELIMINARIES

The controller described below is based on semantic models of the user and its environment. For our system we currently employ formal action descriptions and task models which are compiled into a probabilistic model. All necessary concepts are briefly introduced below.

Hidden Markov Models (HMMs) [7] are probabilistic models, that allow to infer a state of a system that is not observable directly, but through noisy or ambiguous sensor data. An HMM defines a probabilistic model, that consists of a finite number of states, each containing a probability distribution function over sensor observation, that allow to conclude the system state given sensor data. To describe temporal behavior of a system an HMM specifies probabilities for state transitions. HMMs are state of the art methods for activity recognition.

Depending on the available sensors, we can detect the current activity of users. For example, an indoor positioning system can be used to detect whether a user is entering the room and heading for the presentation stage. As customary in activity and intention recognition, we use probabilistic models. Such models can cope with noisy and contradictory sensor data and allow nonetheless to infer the most likely sequence of actions or complex intention. Here, we use Dynamic Bayesian Networks [6], such as HMM's for prob-

SEMAIS'11, Feb 13 2011, Palo Alto, CA, USA

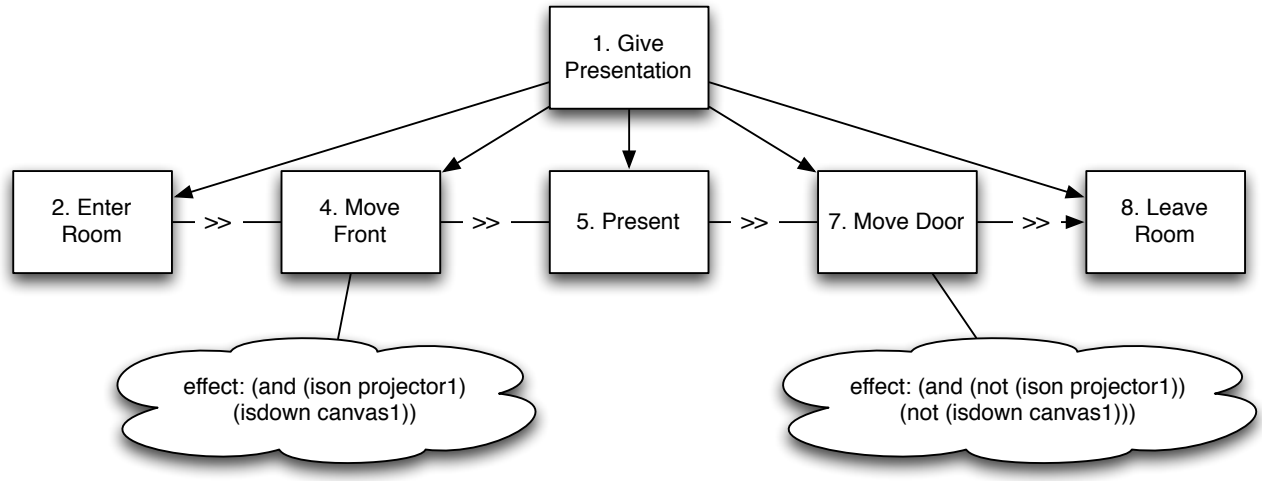

Figure 1: Simplified CTML model describing a typical presentation within our smart environment

abilistic modeling. Calculating a probability distribution over the current state with respect to the observed sensor data as well as the previous state is known as filtering. Doing this requires a model, that describes both, the behavior of the user and the sensor data observable. In addition of recognizing the activity these methods allow to predict future activities, in this case intentions, of the user.

Complex behaviors of (groups of) users can formally be described using CTTE [5] or CTML-models [9], which basically are a hierarchical description of tasks. Sub-tasks can be set into a temporal relation of each other. CTML utilizes temporal operators as the sequence operator (»), the order independence operator (|=|), the concurrent operators (|||) and others that are not used with the examples in this paper. The Collaborative Task Modeling Language (CTML) is especially designed for smart environments and offers features for team modeling, location modeling, device modeling and domain modeling. As described in section 4, we can transfer such a description into a probabilistic model allowing to recognize the current complex action, and thus allowing to infer the overall intention of a sequence of actions.

In the planning domain definition language (PDDL) [8], device actions are formalized as 4-tuples: ⟨Name, Parameters, Preconditions, Effects⟩. Based on such a formal description we can use standard AI planning techniques to infer a plan (sequence of actions) leading from the current to the desired state of the world. Figure 2 and 3 show examples for PDDL descriptions.

3. AN APPLICATION EXAMPLE

The environment where most of our experiments take place is the so called *Smart Appliance Lab*. This room is instrumented with various sensors such as the location tracking system Ubisense [1]. Thus, the location of different users is given by the environment. Other parts of our experimental environment are actuators such as projectors and canvases. Here both sensors and actuators are called devices. Software counterparts of all these devices are provided by the middleware implemented for this environment. These software devices enable us to gain the status of each device inside the room to create a world state. The world state of our environment is thus comprised of the sensor observations and the device states. The environment as well as the middleware controlling the devices of the environment are described in [3]

Projector1 on Canvas1 down	true	false
true	TT	TF
false	FT	FF

Table 1: The cartesian product of all device states forming the world state.

Since the experimental environment may be used as smart meeting room, a typical application is *giving a presentation*. In this scenario the user first enters the room. For our example we assume that the room contains one projector and one canvas. After the user moves to the front of the room where the canvas is located, he prepares the environment for his presentation. Therefore he has to plug in the notebook, set up the projector and lower the canvas. After this is done the user starts his talk and finishes it by moving to the door. Finally the user leaves the room. This example is kept simple to illustrate the main points. The real environment is comprised of eight projectors and eight canvases.

The task specification in Figure 1 contains a detailed description of this example. The annotated effects (illustrated as clouds) describe the desired state of the environment for the following sub-tasks. As description language for task models we use CTML, as described in section 2. The graphical representation of the task model omits the description of the observation data and the priority function. The world state in this example is given in Table 1 and only consists of the two devices. Each of them has a binary state, in case of the projector it is either turned on or turned off. The canvas can be up or down.

Our goal is now to build a controller that recognizes the current state of the user and executes corresponding device actions that makes the annotated effects come true. By executing these action sequences that system automatically assists the user in achieving his goals.

SEMAIS'11, Feb 13 2011, Palo Alto, CA, USA

```
(:action canvasdown
    :parameters (?c - canvas) :
    :precondition (not (isdown ?c))
    :effect (isdown ?c ))
```

Figure 2: A PDDL specification of the CanvasDown action.

```
(:action projectoron
    :parameters (?p - projector) :
    :precondition (not (ison ?p))
    :effect (ison ?p))
```

Figure 3: A PDDL specification of the ProjectorOn action.

4. A CONTEXT-AWARE PROACTIVE CONTROLLER

This section explains how to combine formal descriptions of the environment and the user behavior into a purely reactive probabilistic model. First the description of the user is compiled into a probabilistic model allowing to recognize the user's intentions. Then, we enrich this model by annotating the states with actions, executable by the environment. While running the system, and based on the current state of the environment one of the states will be most likely. The actions attached to the state are then simply executed, resulting in a system supporting the user while achieving his high-level goals.

In the following sections we discuss two different possibilities of enriching the model with device actions. The first is to generate different HMM states for each possible world state. The latter is to annotate the corresponding state with sequences of device actions for all possible world states. A plan for the current world state is then accessible by taking the world state as key for a lookup table resulting in the corresponding plan.

4.1 From Symbolic to Probabilistic Models

We start with the *annotated task model* from Figure 1, which consists of a task model and effects annotated to sub-tasks. Each effect is a subset of the *world state*. It consists of the cartesian product of all device states. We apply the transformation given in [4]. This is done by parsing the syntax tree of the annotated task model and applying the inference rules for each of the temporal operators. The states of the resulting *annotated HMM* correspond to tasks of the task model with corresponding effects. The whole model captures all possible (with respect to the task model) sequences to complete the root task.

We extend the original task model by annotating tasks with their effects with respect to the world state. The effect of the annotated task should be true after executing the task. This can be done by either the user or the controller. Figure 1 contains the additional effect specification for the *Move Front* and the *Move Door* subtasks. In our scenario the effect of the state *Move Front* is that the environment is prepared for the presentation, namely the canvas is down and the projector is on.

To ensure the effects of an annotated HMM state become true, the controller has to execute device actions depending on the current world state. The canvas has to be lowered if it is up but if the projector is already turned on we can omit turning on the projector. Therefore we have to generate sequences of devices actions for each possible situation. This is done by taking each world state as start situation for a planner and the desired subset of the world state, described by the effects of the annotated HMM state as goal. To realize this the planner takes device action specifications, as shown in Figure 2 and Figure 3. Result of this planning step is a sequence of device actions for each possible world state that has to be executed to create the effects specified in the original annotated task model in Figure 1.

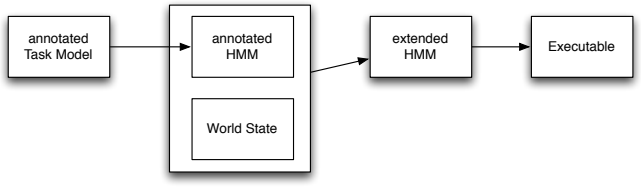

Figure 5: The workflow for creating the controller.

The next two sections describe how to use these world state device action sequence pairs to generate the controller. Both approaches follow the workflow given in Figure 5.

4.2 Unfolding HMM states

In order to create the distinction of the different world states as HMM states, it is necessary to unfold annotated HMM states by using the different world states. Therefore we replace the annotated HMM state by *extended HMM states* that are generated from each possible world state and the state itself. In our example *Move Front* will be replaced by each element of the cartesian product of the world state to ensure that each observation of a world state corresponds to one HMM state. Here *Move Front* is replaced by four new HMM states, each representing a possible world state. Only the HMM state that covers the complete effects has a transition to the HMM state generated from the following sub-task. In our example only the *Move Front TT* state, that assumes that the projector is on and the canvas is down has this transition.

This allows to attach plans to the states in the probabilistic model that needs to be executed in that state as follows: Every annotated user state is combined with every world state, that is a combination of all states of the devices. This world state is used as precondition for device actions during compilation process.

Figure 4 contains the extended HMM that was generated from the task model in Figure 1 combined with the world state defined in Table 1. Please note that self-transitions as well as probabilities for transitions or observations are omitted in the graphical representation.

The generated HMM contains so called slices, that consists of all states generated from one sub-task from the CTML specification. One slice itself was created from the cartesian product of the world state. The intra-slice states differ from each other only by the possible observations of the world state. The names of the intra-slice states illustrated in Figure 4 contain the true/false value given in Table 1. Every state of the slice has an incoming intra-slice transition with a probability given by the number of states. Only the states that create the effects given in task model description have a outgoing inter-slice transition with very high probability. It is possible

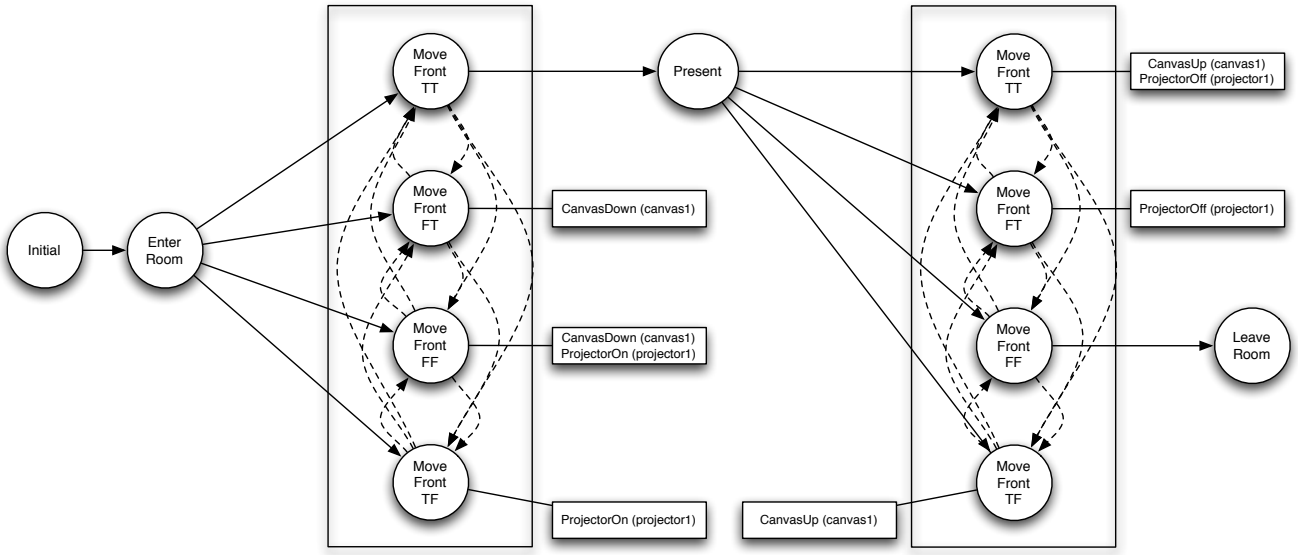

Figure 4: The extended HMM created from the task model and the world state.

that there are more than one state that covers the same effect due to missing effects to single devices of the world. The probabilities of the intra-slice transitions are just given by the number of target states. Inter-slice transitions are generated with respect to the temporal operator of the sub-tasks. The probability of transitions are generated by the normalized weight of the single sub-tasks.

4.3 Lookup table

Another approach to enrich the HMM with device action sequences for user assistance is to create a lookup table for necessary device actions and attach it to the corresponding annotated HMM state. As in the first approach the device action sequences depend on the current world state and need to be generated by a planner that uses the specified effects as goals. Attaching this device action sequence - world state pairs to the annotated HMM state provides a lookup table at runtime. Using the world state, consisting of all device states the table provides the pre-generated device action sequence that has to be executed in order to make the specified effects to the environment become true.

In our scenario the states *Move Front* and *Move Door* are extended by lookup tables for user assistance. The table annotated to the *Move Front* state contains device action sequences that ensure that the projector is turned on and the canvas is down. The *Move Door* HMM state is annotated with a table of device action sequences that ensure that given any world state the projector is turned off and the canvas is up. Figure 6 contains a graphical representation of the generated extended HMM. Probability distribution functions as well as state transition probabilities are omitted for reasons of clarity.

4.4 Choosing one method

The previous sections describe two approaches to attach pre-generated plans to states of a probabilistic model, namely an HMM. Both approaches generate realtime capable systems, that do not have to solve planning problems such as finding a sequence of device actions to support the user. The first approach creates an HMM with

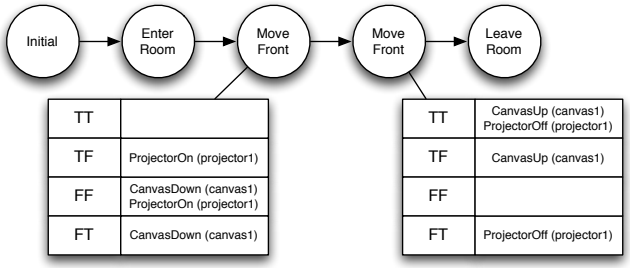

Figure 6: The generated HMM extended by lookup tables for the plans.

very much states, because it creates states for every possible world state. Here the distinction of the world state is done at the state level. The idea of unfolding HMM states by using every possible world state is appropriate if the state of a device is not reliable or noisy.

The second approach is to move the distinction of the world states from different HMM states with attached plans to one HMM state that contains a table of multiple plans, one for each possible world state. The number of HMM states is independent from the world state, which avoids a very high number of states. This approach is applicable whenever the world state is known definitely. This means that each device state is observable without any noise or inconsistency.

5. THE EXECUTION ENVIRONMENT

We developed an execution framework for Bayesian inference that is able to perform fast online filtering of HMM's and particle filters. By separating the model description from the implementation of the algorithms we designed a highly reusable framework. This framework enables users to embed parameterized filters into different environments. This enables us to integrate the generated con-

troller into the software structure described above. Users of this environment only need to implement problem specific details.

A probabilistic model for filtering in our framework has to be implemented in C++. One has to describe three different parts. First a specification of the state space, that is in the case of an HMM represented by a set of states. Second the transition probabilities from each state to another, represented as matrix of probabilities. Third a probability distribution of sensor observations for each state. To provide a more intuitive tool for describing HMM's we introduced a description language that supports a simple way to describe HMM based models.

The compilation process creates the state space of our controller from the single sub-tasks of the user model combined with each possible device state combination. The transition probabilities are given by the probabilities of the model generated only by the subtasks as described in [4] and the observation probability distributions of the original approach are extended by an observation of the device states in the extended state. A model specified in the way needs sensor data and the world state as input and provides a sequence of device actions as output. These device actions need to be executed in order to support the user.

6. SUMMARY AND OPEN PROBLEMS

In this paper we showed that a context-aware controller for smart environments can be created from a combination of semantic models of the user and the environment. The controller is based on bayesian inference where the model was generated from task-based specification of users together with a precondition and effect specifications of each device forming the environment. Sensor data as well a accumulated world state serve as input, a device action sequence, that needs to be executed as output of the inference process. We introduced two ideas to merge task based user models and precondition and effects specification of the environment to create probabilistic models that assist the user.

Further research should include smart environment evaluation of the controller described here. Both approaches should be evaluated and the results should be compared for different devices and scenarios. This includes tests for maximum manageable complexity of the state space as well as minimal complexity that creates sufficient user support. Due to the compile time planning process we are able to pre-generate action sequences. This allows to find modeling problems such as deadlocks at compile time. Our approach in this paper utilizes HMM's for inference. However, since the state space may explode and exact inference will not be suitable, we can change the inference algorithm to Monte Carlo based methods such as particle filters. These methods are already supported by the execution environment described above.

The controller introduced here is described as central service. It is possible to decentralize this approach to the usage of multiple services, each of them describing a subspace of the model. By comparing the likelihood of multiple services, it is be possible to choose either an action sequence of one agent or a combination of multiple sequences that do not disturb each other.

Another point that should be analyzed is how both systems behave if only the most probable device action sequences will be preplanned. If the system reaches a state that does not contain a device action sequence the plan has to created at runtime. The realtime behavior of this extension has to be examined.

Acknowledgements

Frank Krüger's work in the MAXIMA project as well as Gernot Ruschers's work in the MAIKE project are both supported by *Wirtschaftsministerium M-V* at expense of *EFRE* and *ESF*.

7. REFERENCES

[1] http://www.ubisense.de, JUN 2009.

[2] *Ubicomp '10: Proceedings of the 12th ACM international conference on Ubiquitous computing*, New York, NY, USA, 2010. ACM. 608109.

[3] S. Bader, G. Ruscher, and T. Kirste. Decoupling smart environments. In S. Bader, T. Kirste, W. G. Griswold, and A. Martens, editors, *Proceedings of PerEd2010*, Copenhagen, SEP 2010.

[4] C. Burghardt, M. Wurdel, S. Bader, G. Ruscher, and T. Kirste. Synthesising generative probabilistic models for high-level activity recognition. In *Activity Recognition in Pervasive Intelligent Environments*. Atlantis Press, Paris, France, 2010. To appear.

[5] G. Mori, F. Paterno, and C. Santoro. Ctte: Support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering*, 28:797–813, 2002.

[6] K. P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley, CA, USA, 2002.

[7] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[8] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2009.

[9] M. Wurdel, D. Sinnig, and P. Forbrig. CTML: Domain and Task Modeling for Collaborative Environments. *J. UCS*, 14(19):3188–3201, 2008.