Deterministic and Probabilistic Implementation of Context

Oliver Brdiczka, Patrick Reignier, James L. Crowley, Dominique Vaufreydaz, Jérôme Maisonnasse
Laboratoire GRAVIR
INRIA Rhône-Alpes
655 Av. de l'Europe
38330 Montbonnot, France
{brdiczka, reignier, crowley, vaufreydaz, maisonnasse}@inrialpes.fr

Abstract

This paper addresses the problem of implementing an abstract context model. First, the abstract context model is represented by a network of situations. Two different implementations for the situation model are then proposed: a deterministic one based on Petri nets and a probabilistic one based on Hidden Markov Models. Both implementations are illustrated and applied to real-world problems.

1. Introduction

Pervasive and ubiquitous computing [16] integrate computation into all-day environments. People are enabled to move around and interact with computers more and more naturally. One of the goals of these computerized spaces is to enable devices to sense changes in the environment and to automatically adapt and act based on these changes. A main focus is laid on sensing and responding to human activity.

Human activity does not strictly follow plans but is very situation dependent [13]. Computerized spaces and their devices need hence to use this situational information, i.e. *context* [4], to respond correctly to human activity. In order to become context-aware, computer systems must maintain a model describing the environment, its occupants and their activities.

In this paper, we describe how an abstract context model [3] based on the notion of situation [4] can be represented by different implementations. A probabilistic implementation based on Hidden Markov Models and a deterministic implementation based on Petri Nets is discussed. Both implementations have been applied to real-world problems (given as examples).

2. Representing abstract context by situation models

The notion of context is not new and has been explored in different areas like linguistics, natural language processing and knowledge representation. Dey defines context as "any information that can be used to characterize the situation of an entity" [4]. An entity can be a person, place or object considered relevant to user and application. Context-aware applications need this contextual information to deliver the correct service to the correct user, at the correct place and time, and in the correct format for the environment [16]. The structure and representation of this information must be determined before being exploited by a specific application.

Context and activity are separable. The context describes features of the environment *within which* the activity takes place [5]. Loke states that situation and activity are, however, not interchangeable, and activity can be considered as a type of contextual information which can be used to *characterize* a situation [8].

Following Dey's context definition, situation is a central notion describing context. Crowley et al. consider context to be a network of situations [3]. Dey defines situation as "description of the states of relevant entities" [4]. Situation is thus a *temporal state* within context. Allen's temporal operators [1] can be used to describe relationships between situations.

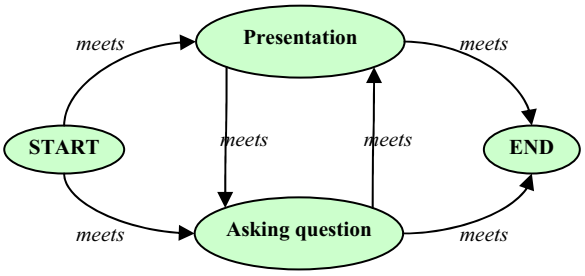


Figure 1 Context of a "presentation with questions". The conceptual events characterizing the situations are not detailed.

There are different concepts used to *characterize* a situation. As described above, activity is such a concept. Activity models like in [9] identify the different states of entities. Crowley et al. extend this concept to roles and relations between entities [3]. An entity is observed to play a role if it passes a role acceptance test on its properties. A relation is defined as a predicate function on

Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06)
0-7695-2520-2/06 $20.00 © 2006 IEEE

several entities playing roles. Changes in activity, role or relation can be signaled by *events*.

Behavior within the environment can be described by a *script*. A script corresponds to a sequence of situations in the situation network reflecting (human) behavior in the environment. Scripts are not necessarily linear.

To summarize, we describe context as a *situation model*. The situations are in temporal relationship within a network. Changes between the states of context, i.e. the situations, are activated by *events* referring to concepts describing situation. Behavior within the environment can be described by *scripts*. Figure 1 gives a simple example of a context.

3. Implementing situation models

Situations and the underlying abstract concepts can be interpreted as finite-state machines. The finite-state machine implementation influences the control flow and how perceptions, coded as events, are finally used and interpreted to activate situations. We present a deterministic and a probabilistic implementation of situation models. The deterministic implementation is based on Petri Nets, while the probabilistic implementation is based on Hidden Markov Models. The choice of the implementation depends on the application that is envisaged. An example is given for each implementation in the following sections.

3.1 Deterministic implementation: Petri Nets

We begin this section with an informal review of Petri Nets. We then describe how Petri nets are used to implement a network of situations within a context and how to evaluate a script.

3.1.1. Petri Nets. A Petri net is a graphical mathematical tool used to model dynamical systems with discrete inputs, outputs and states. Such models have first been defined by C. M. Petri [11]. A Petri Net is an oriented graph, composed of arcs and two categories of nodes (places and transitions). Places are the state variables of the system containing zero or a positive number of marks. Transitions model the system evolution and are connected *from* places *to* places. A transition is valid if all the "from" places have at least one mark. When a valid transition is fired: one mark is removed from every "from" place and one mark is added to every "to" place. Only one transition can be fired at a time. A more formal definition of a Petri net is given in [10].

Finite-state machines like situation models can be equivalently represented by a subclass of Petri nets [10]. Several extensions of the Petri Net model have been proposed. One of them is the *synchronized Petri net*. A synchronized Petri net is a Petri Net with *events* associated to each transition. A transition can now be fired if it is valid and the corresponding event has been triggered (Figure 2).

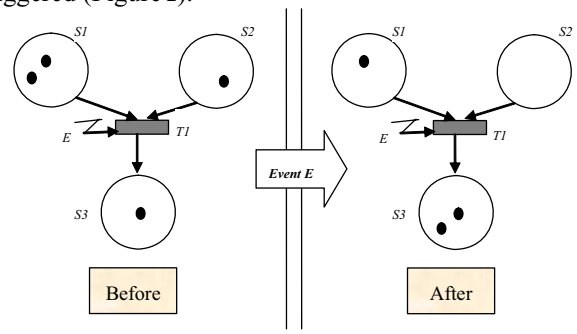
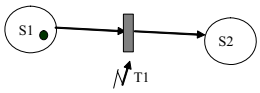
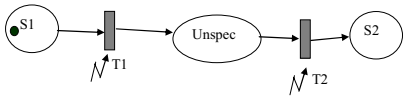
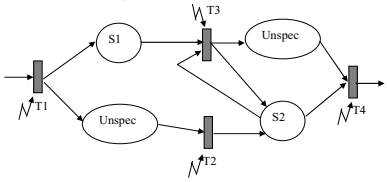
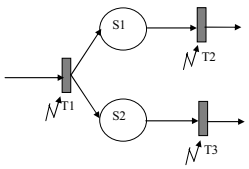
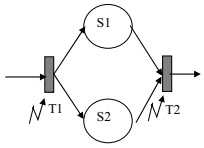
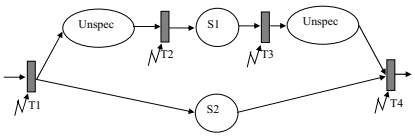
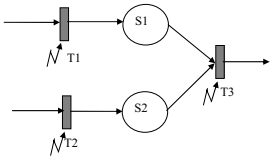


Figure 2 Synchronized Petri net with places S1, S2, S3 and transition T1 triggered by Event E.

3.1.2. Implementation and script evaluation using Petri nets. A context is defined by a network of situations. The arcs between the situations are temporal constraints based on the Allen temporal operators [1]. *Events* indicate state changes of the concepts (activity, role, relation or others) describing situations. A situation S is validated by the set *ValidS* of relevant events. The event condition for a transition *T* can be described using relevant event sets of places (situations) before and after the transition *T*. The following table shows the transformations of the Allen operators between situations into the corresponding synchronized Petri nets.

Allen Operator	Synchronized Petri net + Event conditions
S1 meets S2	 $T1 = \neg ValidS1 \wedge ValidS2$
S1 before S2 S2 after S1	 $T1 = \neg ValidS1,\ T2 = ValidS2$
S1 overlaps S2	 $T1 = ValidS1,\ T2 = ValidS2,$ $T3 = \neg ValidS1,\ T4 = \neg ValidS2$
S1 starts S2	

	$T1 = ValidS1 \wedge ValidS2,$ $T2 = \neg ValidS1,\ T3 = \neg ValidS2$
S1 equals S2	 $T1 = ValidS1 \wedge ValidS2$ $T2 = \neg ValidS1 \wedge \neg ValidS2$
S1 during S2	 $T1 = ValidS2,\ T2 = ValidS1,$ $T3 = \neg ValidS1,\ T4 = \neg ValidS2$
S1 finishes S2	 $T1 = ValidS1,\ T2 = ValidS2,$ $T3 = \neg ValidS1 \wedge \neg ValidS2$

Using this table, we can thus transform a situation network into a corresponding synchronized Petri net.

3.1.3. Example: Intelligent cameraman [6]. The intelligent cameraman is system that automatically records a lecture. The lecture room is equipped with multiple cameras and microphones. The system is context-aware selecting at every time, based on the current situation, the appropriate camera to provide images.

Figure 3 Different camera images recorded by the intelligent cameraman system.

The lecture is an alternation of "lecturer speaking" and "audience asking a question". "New slide" and "Someone entering the room" can happen in parallel. The situation network is given in Figure 4, the corresponding Petri Net is given in Figure 5.

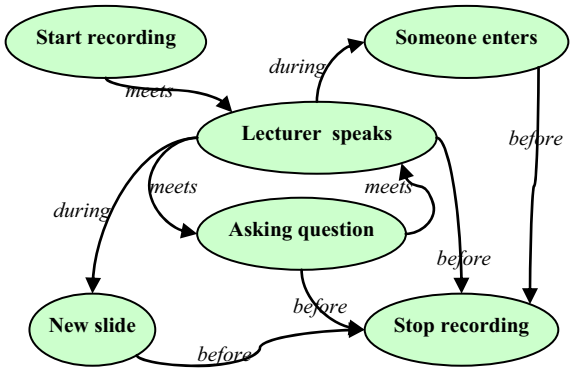


Figure 4 Situation network of intelligent cameraman system.

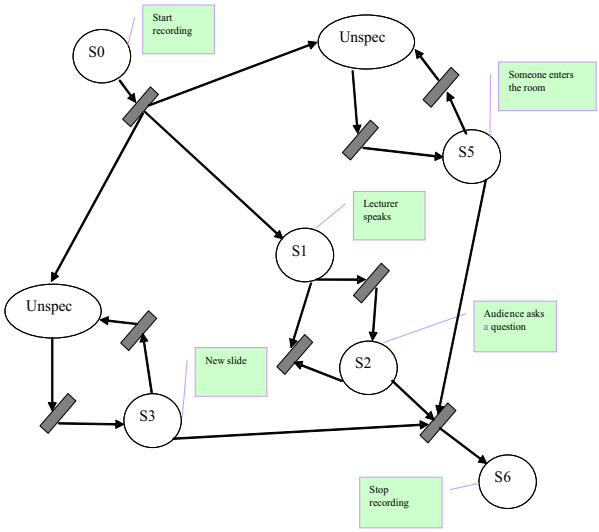


Figure 5 Petri net of intelligent cameraman system.

Code generation is done by automatically transforming the synchronized Petri net into a corresponding program in JESS [7]. JESS is an expert system programming environment (facts database plus forward chaining rules). The input of the generated program are facts based on events describing state changes of the concepts (activity, role, relation or others). The output are the current situation(s) and the associated action(s).

Note that Petri nets are very well adapted for implementing situation models containing parallelism. Petri nets are, however, less suitable for applications with erroneous perceptions or uncertain perception expectations.

3.2 Probabilistic implementation: Hidden Markov Models

A probabilistic implementation of the situation model integrates uncertainty values into the model. These uncertainty values can both refer to confidence values for events and to a less rigid representation of situation and situation transitions.

The situation model is a finite-state machine. The natural choice for a probabilistic implementation is then a probabilistic finite-state machine [14]. A probabilistic finite-state machine is a probabilistic automaton (PFA) defined over a finite alphabet Σ. A language is a subset of Σ*. A PFA defines a stochastic language, which is a probability distribution over Σ*. The distribution must verify $\sum_{x\in\Sigma^*} \Pr obability(x) = 1$. A formal definition of PFA can be found in [14].

3.2.1. Hidden Markov Models. A Hidden Markov Model [12] is a stochastic process where the evolution is managed by states. The series of states constitute a Markov chain which is not directly observable. Such a chain is said to be "hidden". Each state of the model generates an observation. Only the observations are visible. The objective is to derive the state sequence and its probability, given a particular sequence of observations. A more formal definition of an HMM is given in [12]. The two following propositions hold [15]:

Proposition 1: Given a PFA *A* with m transitions and *Probability(epsilon)* = 0, there exists an HMM *M* with at most m states such that the stochastic language D_M of *M* is equal to the stochastic language D_A of *A*.

Proposition 2: Given an HMM *M* with n states, there exists a PFA *A* with at most n states such that the stochastic language D_A of *A* is equal to the stochastic language D_M of *M*.

As we are only interested in PFA without epsilon transitions, i.e. PFA the transitions of which are triggered by events, language-equivalent HMMs can be used to implement PFA.

3.2.2. Implementation and script evaluation using Hidden Markov Models. The situations of a context model can be implemented by the states of a HMM. *Events* indicating state changes of the concepts (activity, role, relation or others) generate the observations for the HMM. A state (situation) is characterized by a particular probability distribution of these observations. The activation of a new situation (state) is thus determined by the transition probability from the current state to this new state as well as by the probability of the given observations in this new state. The connections in the situation network are represented by non-zero transition probability values. The observation probability distributions for the situations as well as the transition probabilities between the situations need to be specified (or learned) when implementing a situation model. We are interested in three basic problems:

1. Given a sequence of observations (based on events) and a situation model implemented by a HMM, how to choose the corresponding state sequence (situation sequence)? This includes the determination of the (most likely) current situation and the determination of likely following situations.

2. Given a sequence of observations (based events) and a situation model implemented by a HMM, how to compute the probability of the observation sequence, given the model? This corresponds to the likelihood of the situation model (based on the given observations).

3. How to adjust the HMM model parameters? This corresponds to adjusting probability distributions based on given data.

[12] gives several solutions to these problems. The Viterbi algorithm is used to determine the most probable state sequence, given a HMM and an observation sequence (Problem 1). The probability of a HMM, given an observation sequence, can be computed using the Forward-Backward algorithm (Problem 2). The expectation-maximization (EM) Baum-Welch algorithm adjusts the HMM model parameters, given observation sequences (Problem 3).

Note that the HMM implementation of a situation model is particularly suitable for applications that deal with erroneous perceptions as well as situations that are characterized by a particular frequency of events. A HMM implementation is, however, less suitable for representing parallelism (not all Allen operators can thus be represented by a classical HMM).

3.2.3. Example: Detection of Interaction Groups [2]. This example addresses the problem of detecting interaction groups in an intelligent environment. The dynamic change of interaction group configuration, i.e. the split and merge of interaction groups, can be seen as indicator of new activities. Our goal is to determine the current small group configuration from speech activity event data. We focus thus on verbal interaction, which further implies a minimum size of two individuals for one group (assuming that isolated individuals do not speak).

The proposed approach is based on a HMM implementation of the context model. The observations of

the HMM are a discretization of speech activity events sent by an automatic speech detector [6]. This detector parses multi-channel audio input and detects which individual stops and starts speaking.

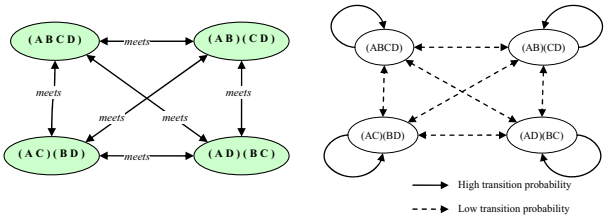


Figure 6 Situation model (left) and states of HMM implementation (right) for a meeting of 4 individuals A, B, C, D.

Figure 6 shows the situation model and the corresponding HMM for a meeting with 4 individuals. Each possible group configuration is represented by a situation. The probability distributions of the different situations are specified based on conversational hypotheses [2]. The transition probabilities between the states are set to a very low level in order to stabilize the detection of state changes assuming hence that group changes occur in reasonable delays. To detect different group configurations, we apply the Viterbi algorithm (solution to Problem 1 in section 3.2.2) to the flow of arriving observations.

Figure 7 Example configuration of 2 groups of 2 individuals.

To evaluate, we recorded the interactions of 4 individuals during 3 experiments (Figure 7). The speech was recorded using lapel microphones. We obtain a total recognition rate for the group configurations of 84.8 % [2].

4. Conclusion

This paper proposed two different implementations for the situation model representing abstract context: a deterministic one based on Petri nets and a probabilistic one based on Hidden Markov Models. Both implementations have been applied to real world problems with success: an intelligent cameraman system (Petri nets) and an interaction group detector (HMMs) have been implemented. Each implementation is well adapted for particular applications: Petri nets for parallelism and Hidden Markov Models for erroneous or uncertain input.

5. References

[1] Allen, J., *Maintaining Knowledge about Temporal Intervals*, Comm. ACM, 26 (11):832-843, 1983.

[2] Brdiczka, O., Maisonnasse, J., and Reignier, P., *Automatic detection of interaction groups.* Proc. of ICMI, October 2005.

[3] Crowley, J.L., Coutaz, J., Rey, G. and Reignier, P., *Perceptual Components for Context Aware Computing*, Proc. of UbiComp, 2002.

[4] Dey, A.K., *Understanding and Using Context*, Personal and Ubiquitous Computing 5:4-7, 2001.

[5] Dourish, P., *What we talk about when we talk about context*, Personal and Ubiquitous Computing 8:19-30, 2004.

[6] *FAME: Facilitating Agent for Multi-Cultural Exchange (WP4)*, European Commission project IST-2000-28323 October 2001.

[7] Jess: the rule engine for java, http://herzberg.ca.sandia.gov/jess/.

[8] Loke, S.W., *Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective*, The Knowledge Engineering Review, 19(3):213-233, 2005.

[9] Muehlenbrock, M.; Brdiczka, O.; Snowdon, D., Meunier, J., *Learning to Detect User Activity and Availability from a Variety of Sensor Data*, Proc. of PerCom, 2004.

[10] Murata, T., *Petri Nets: Properties, Analysis and Applications*, Proc. of IEEE 77(4):541-580, 1989.

[11] Petri, C. A., *Kommunikation mit Automaten,* Ph.D. thesis, Institut fuer Instrumentelle Mathematik, Bonn, 1962.

[12] Rabiner, L., *A tutorial on Hidden Markov Models and selected applications in speech recognition*, Proc. of IEEE 77(2):257-286, 1989.

[13] Suchman, L., *Plans and Situated Actions: The Problem of Human-Machine Communication*, Cambridge University Press, 1987.

[14] Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, and Carrasco, R. C., *Probabilistic Finite-State Machines – Part I.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005.

[15] Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, and Carrasco, R. C., *Probabilistic Finite-State Machines – Part II.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005.

[16] Weiser, M., *The Computer for the Twenty-First Century*, Scientific American Publishers, 1991.