Context Aware Environments : from Specification to Implementation

- **Authors :** Patrick Reignier, Oliver Brdiczka, Dominique Vaufreydaz, James L Crowley, Jerôme Maisonnasse

- **Keywords :** Context recognition, Petri nets, hidden Markov models, automatic cameraman, interaction group detection

- **Authors Address :** Inria Rhône-Alpes, 655 avenue de l'Europe, 38334 Saint Ismier Cedex, France
- **Contact :** Patrick Reignier
- **Contact's email :** Patrick.Reignier@inrialpes.fr
- **Contact's phone number :** +33 476 615 411
- **Contact's fax number :** +33 476 615 210

Context Aware Environments : from Specification to Implementation

Patrick Reignier, Oliver Brdiczka, Dominique Vaufreydaz
James L Crowley, Jerôme Maisonnasse

Inria Rhône-Alpes, 655 avenue de l'Europe, 38334 Saint Ismier Cedex, France
{reignier,brdiczka,vaufreydaz,crowley,maisonnasse}@inrialpes.fr

Abstract

This article deals with the problem of implementing a context model for a smart environment. This problem has already been addressed several times using many different data or problem driven methods. In order to separate modeling phase from implementation, we first represent the context model by a network of situations. Then, different implementations can be automatically generated from this context model depending on user needs and underlying perceptual components. Two different implementations are proposed in this paper: 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.

Keywords : *Context recognition, Petri nets, hidden Markov models, automatic cameraman, interaction group detection*

1 Introduction

Pervasive and ubiquitous computing [Weiser(1991)] seek to integrate computation into everyday 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 [Suchman(1987)]. Computerized spaces and their devices need hence to use this situational information, i.e. *con-*

text [Dey(2001)] , 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.

Unfortunately, there is no generic algorithm capable of robustly recognizing situations from perceptual events coming from sensors. Various approaches, based on logic, Bayesian rule, fuzzy logic etc., have been explored and evaluated. However, their performance is very "problem[1] and environment dependent". In order to be able to use several approaches inside the same application, we have decided to clearly separate the specification of context (scenario) and the implementation of the program that recognizes it. The transformation between a specification and its implementation must be as automatic as possible.

In this article, we first review basic concepts for modelling context. The central notion of situation [Dey(2001)] is defined and context is represented by a network of situations [Crowley(2002)]. Situations within this network are in temporal relationships (described by Allen's temporal operators [Allen(1983)]). This is the formal specification of the context. We then propose two different approaches for implementing it : a deterministic implementation transforming the situation network into a synchronized Petri net and a probabilistic implementation transforming the situation network into a hidden Markov model. Both implementations have been applied to real world problems. An automatic cameraman system (section 3.3) has been implemented using synchronized Petri nets and a detector for interaction group configurations (section 4.3) has been implemented using hidden Markov models. Both example implementations have been tested with success.

2 Representing 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" [Dey(2001)]. 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 [Weiser(1991)]. The structure and representation of this information must be determined before being exploited by a specific application.

[1]The problem can be to recognize a situation in a well defined scenario (e.g. "presentation" or "brainstorming" in a meeting scenario) or to recognize the scenario within a set of possible scenarios (e.g. is it a meeting, party or candlelight dinner?)

Context and activity are separable. The context describes features of the environment *within which* the activity takes place [Dourish(2004)]. 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 [Loke(2005)].

Following Dey's context definition, situation is a central notion describing context. We consider context to be a network of situations [Crowley(2002)]. Dey defines situation as "description of the states of relevant entities" [Dey(2001)]. Situation is thus a *temporal state* within context. Allen's temporal operators [Allen(1983)] can be used to describe relationships between situations (Fig. 1).

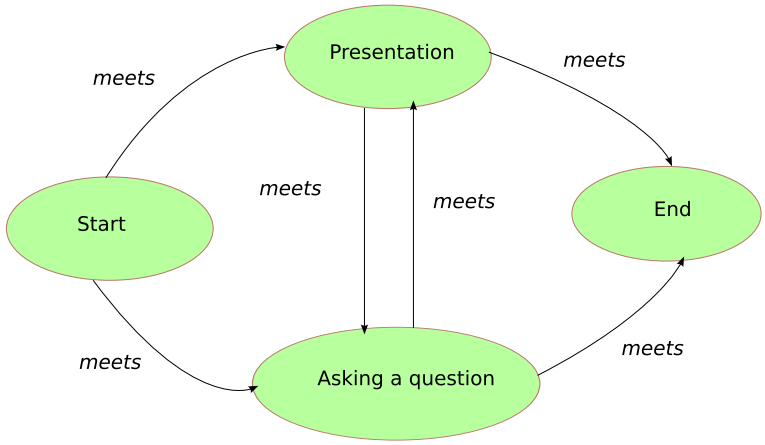


Figure 1: Context of a "presentation with questions". The conceptual events characterizing the situations are not detailed.

There are different concepts used to *characterize* a situation. As described above, activity is such a concept. Activity models like in [Muehlenbrock(2004)] identify the different states of entities. We extend this concept to roles and relations between entities [Crowley(2002)]. 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 several entities playing roles. Changes in activity, role or relation are 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.

Situations and the underlying concepts (activity, roles or relations) are the context meta-model. This meta-model can be used by the software developer to specify the situations and the associated reactions (the model). This model is only a specification. It must be transformed into a working system listening to perception events, interpreting them and verifying that this flow of events is

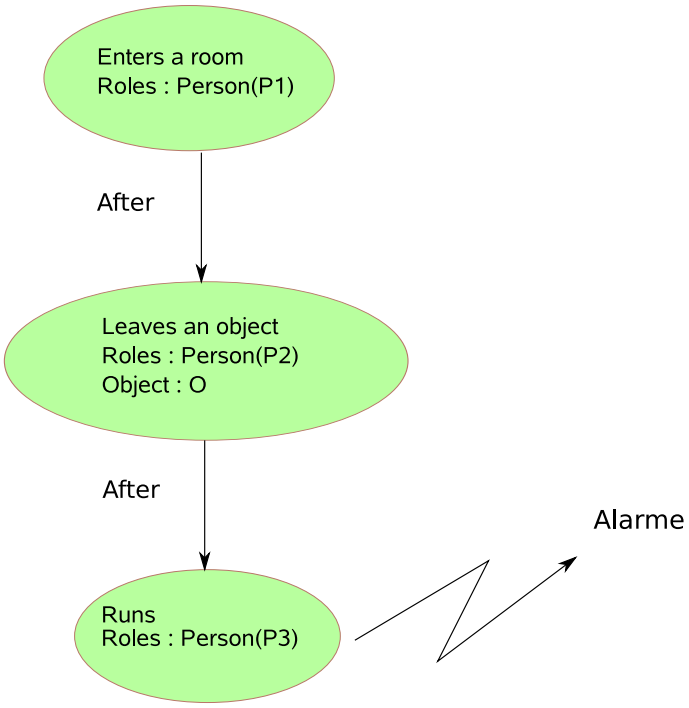


Figure 2: The "leaving an object" context specification

compatible with the model specification.

Let us consider for instance a "leaving object" context. An alarm must be triggered when someone enters a room, leaves an object and run away. It can be specified by the model figure 2. The perception events can be provided by a video tracker. The *Entering a room* event is triggered when a moving target goes through a detection zone in front of the door. The *Leaving object* event is detected by a target split. The *run event* is triggered by a moving target with a speed higher than a specified threshold.

Based on this context specification, the next step must be to write a program checking that the *run* event is coming *after* the *leaves an object* event. Unfortunately, there is no generic algorithm capable of robustly recognizing situations from perceptual events coming from sensors. Various approaches have been proposed but they are very problem dependant. We argue that instead of trying to find a "universal solution", we must have an automatic way to transform a context specification into a set of context recognition modules based on various approaches. It is then easier to try several approaches on a particular application. The developer can also modify the context specification without having to recode the various implementations.

In the following sections, 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.

3 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 situation network representing a context model and how to evaluate a script.

3.1 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 [Petri(1962)]. 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 Petri nets is given in [Murata(1989)].

Finite-state machines like situation models can be equivalently represented by a subclass of Petri nets [Murata(1989)]. Several extensions of Petri nets 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 (Fig. 3).

3.2 Implementation and Script Evaluation using Petri Nets

A context model is defined by a network of situations. The connections between the situations are temporal constraints based on Allen's temporal operators [Allen(1983)]. *Events* indicate state changes of the concepts (activity, roles, or relations) describing situations. A situation change is triggered by these events. For instance, consider two situations S1 and S2 such that S2 meets S1 (Fig. 4). To trigger a change from the current situation S1 to situation S2, we need to observe at least two events:

1. a first event invalidating the situation S1 ($\neg ValidS1$), followed by
2. a second event validating situation S2 ($ValidS2$) .

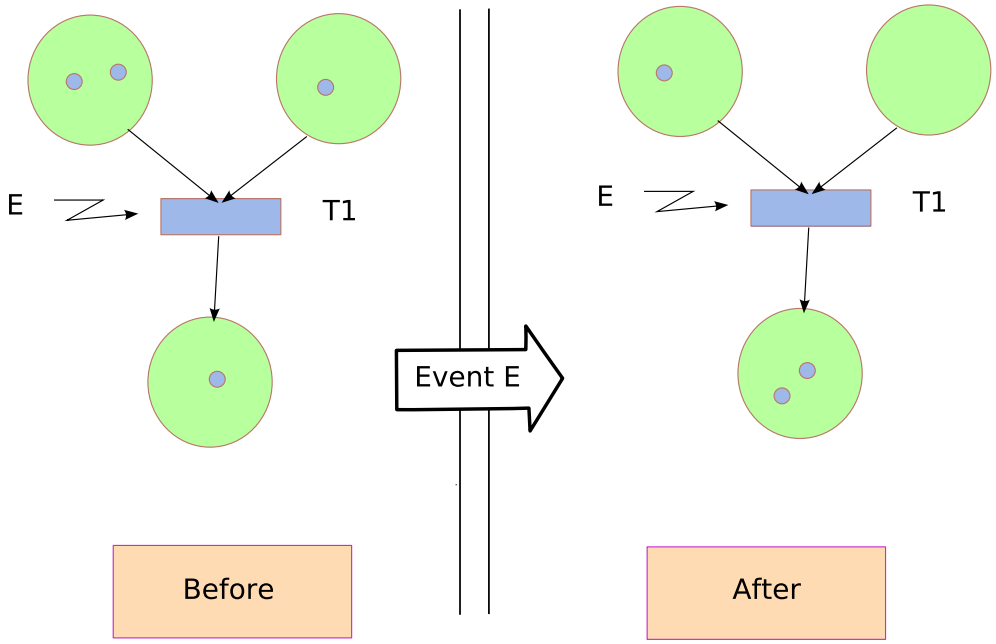


Figure 3: Synchronized Petri net with places S1, S2, S3 and transition T1 triggered by event E.

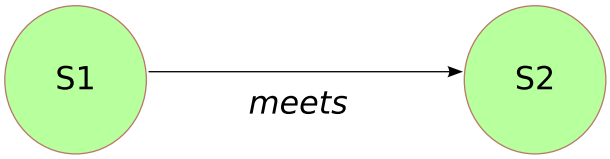


Figure 4: A small context : S2 meets S1

This example can be directly represented as a synchronized Petri net (Fig. 5). We associate a situation to every place. The situation is active if there is at least one mark in the corresponding place: we are currently in situation S1.

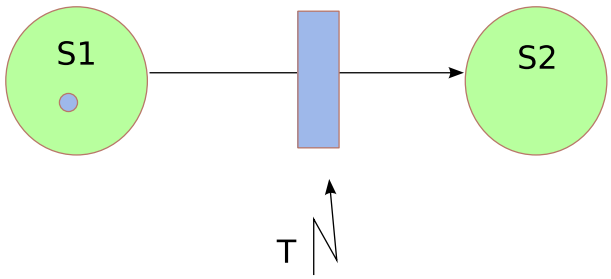


Figure 5: Synchronized Petri net implementation of the context

In a "standard" Petri net, the transition between S1 and S2 is automatically fired as soon as S1 is active. We need to control this transition based on the perceptual events coming from the environment. This transition control is managed using the transition event of the synchronized Petri net. We generate this transition event T only when Situation S1 is no more valid and Situation S2 becomes valid (as seen in the previous paragraph):

$$T = \neg \text{ValidS1} \wedge \text{ValidS2}$$

More generally, the event function T of a transition is the conjunction of a function associated to the place before the transition and a function associated to the place after the transition. We will call those two functions respectively $PreT$ and $PostT$. We have:

$$T = PreT \wedge PostT$$

$PreT$ corresponds to what we are *currently observing* in the environment. $PostT$ indicates what the system *should expect* to see next based on the contextual model.

This short example showed how to transform the "meet" operator into a corresponding synchronized Petri net. This transformation can be done for all the other Allen operators as summarized in Table 1.

Note that Petri nets are very well adapted for implementing situation models containing parallelism. As shown in Table 1, all Allen temporal operators can be implemented by Petri nets. However, Petri nets are not suitable for applications with erroneous perceptions or uncertain

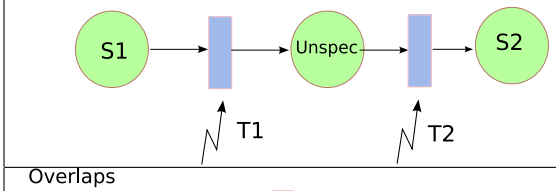
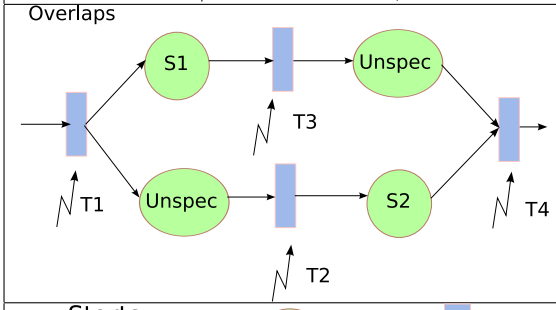
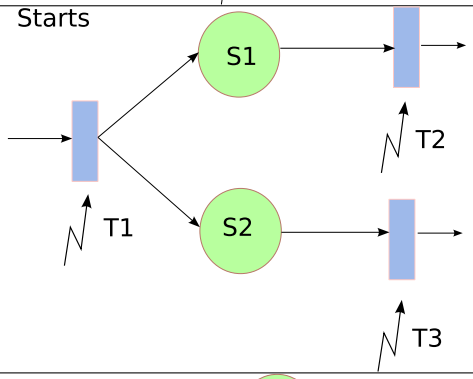
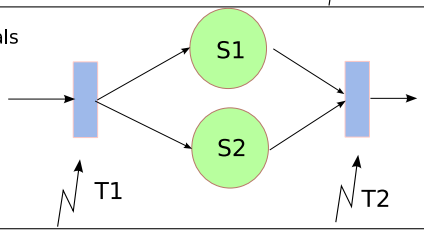
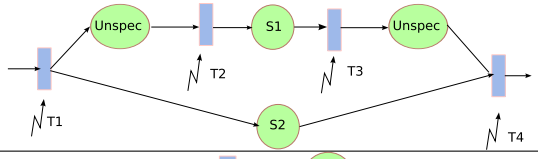
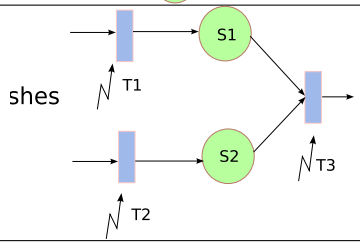
Petri net	Transition function
Before 	$\begin{cases} T1 = \neg ValidS1 \\ T2 = ValidS2 \end{cases}$
Overlaps 	$\begin{cases} PostT1 = ValidS1 \\ T2 = ValidS2 \\ T3 = \neg ValidS1 \\ PreT4 = \neg ValidS2 \end{cases}$
Starts 	$\begin{cases} PostT1 = ValidS1 \wedge ValidS2 \\ PreT2 = \neg ValidS1 \\ PreT3 = \neg ValidS2 \end{cases}$
Equals 	$\begin{cases} PostT1 = ValidS1 \wedge ValidS2 \\ PreT2 = \neg ValidS1 \wedge \neg ValidS2 \end{cases}$
During 	$\begin{cases} PostT1 = ValidS2 \\ T2 = ValidS1 \\ T3 = \neg ValidS1 \\ PreT4 = \neg ValidS2 \end{cases}$
shes 	$\begin{cases} PostT1 = ValidS1 \\ PostT2 = ValidS2 \\ PreT3 = \neg ValidS1 \wedge \neg ValidS2 \end{cases}$

Table 1: Synchronized Petri nets for the Allen operators: the "Unspec" place stands for an unspecified situation, which means that we do not model what might happen.

perception expectations.

3.3 Example : Automatic Cameraman

The automatic cameraman [FAME(2004)] is an audio-video recording 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 (Fig. 6).

Figure 6: Different camera images recorded by the automatic cameraman system.

The available actions are *Start recording*, *Filming the whole room*, *Filming the lecturer*, *Filming the audience* and *Filming the slides*. The situation list is:

- Init → start recording and filming the whole room.
- The lecturer speaks → filming the lecturer.
- Someone in the audience asks a question → filming the audience.
- There is a new slide → filming the new slide.
- Some one enters the room → filming the whole room.

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 Fig. 7, the corresponding Petri net is given in Fig. 8.

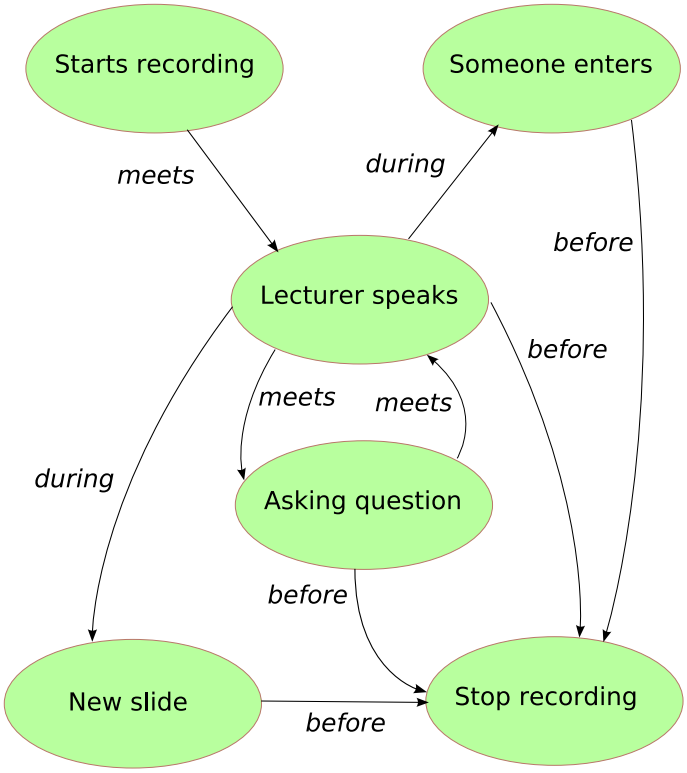


Figure 7: Situation network of the automatic cameraman system.

Code generation is done by automatically transforming the synchronized Petri net into a corresponding program in JESS [Jess]. JESS is an expert system programming environment (facts database with forward chaining rules). The input of the generated program are facts based on events describing state changes of the concepts (roles and relations here)[2]. The output are the current situation(s) and the associated action(s).

We are going to illustrate this code generation process on a small subset of the previous Petri net (Fig. 9).

The generated rules can be decomposed into three groups:

1. The Petri algorithm.
2. The events calculation (the synchronization).
3. The action generation.

[2]Activity, roles, and relations are reconstructed from data provided by perceptual components (video tracker, speech activity detectors, etc.)

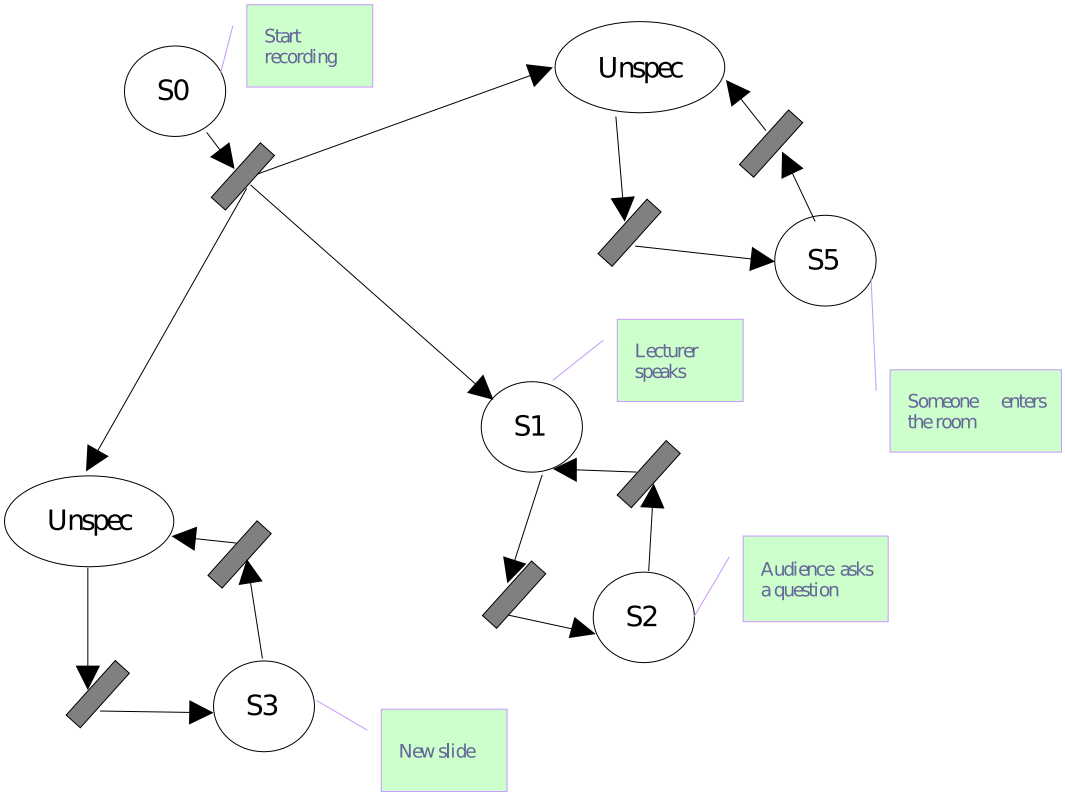


Figure 8: Petri net of the automatic cameraman system.

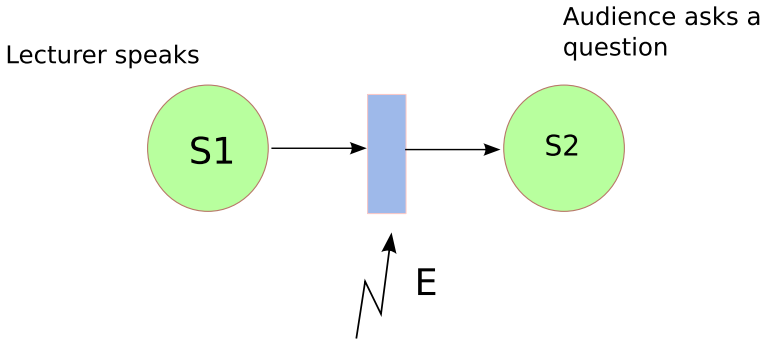


Figure 9: The lecturer speaks and then the audience asks a question

Petri algorithm:

These rules implement the calculation of the marks: if there is at least one mark in place S1 and the associated event is triggered, we remove a mark in S1 and add a new one in S2.

```
(defrule t2
  ?idEvent <- (event (name "t2"))
  ?sit0 <- (situationPlace (name "S1") (mark ?m0&:(neq ?m0 0)))
  ?sit1 <- (situationPlace (name "S2") (mark ?m1))
  =>
  (retract ?idEvent)
  (modify ?sit0 (mark (- ?m0 1)))
  (modify ?sit1 (mark (+ ?m1 1)))
)
```

Events calculation:

The transition event describes that S1 is no longer valid and that S2 has become true. A situation is described here by a set of roles and relations. A situation becomes true if its roles and relations are validated. A situation becomes invalid when there is a change in the assignment of entities to roles or a change in the relations between those entities.

As explained in subsection 3.2, the transition event describes that $S1$ is no longer valid and that $S2$ has become true. A situation is described by a set of roles and relations. A situation becomes true if its roles and relations are validated. A situation becomes invalid when there is a change in the assignment of entities to roles or a change in the relations between those entities.

- Situation S1:
  - Roles : lecturer, speaker.
  - Relations : lecturer *is same as* speaker.
- Situation S2:
  - Roles : audience, speaker.
  - Relations : audience *is same as* speaker.

```
(defrule t2EventTransition
  ?pre_S1 <- (situation  (name "S1") (entities ?lecturer ?speaker))
  (situationPlace (name "S1") (mark ?m_S1&:(neq ?m_S1 0)))
  (or
     (not (lecturer (isPlayedBy ?lecturer)))
     (not (speaker (isPlayedBy ?speaker)))
     (not (isSameAs (isVerifiedBy ?lecturer ?speaker)))
  )
  (audience (isPlayedBy ?new_audience))
  (speaker (isPlayedBy ?new_speaker))
  (isSameAs (isVerifiedBy ?new_audience ?new_speaker))
  ?post_S2 <- (situation (name "S2"))
  =>
  (modify ?tr (lock (- ?l 1)))
  (assert (event (name "t2")))
  (modify ?pre_S1 (entities))
  (modify ?post_S2 (entities ?new_speaker ?new_audience))
)
```

Action generation:

These rules associate (if needed) an action to every situation.

```
(defrule d1
  (situationPlace (name "S1") (mark ?m&:(neq ?m 0)))
  =>
  (assert (askForCamera ?*lecturerCamera*) )
  (trace "Situation lecturer parle")
)

(defrule d2
  (situationPlace (name "S2") (mark ?m&:(neq ?m 0)))
  =>
  (trace "Situation Audience pose une question")
  (assert (askForCamera ?*audienceCamera*) )
)
```

This approach recorded a four day seminar on "Language Technology" and "Language, Cognition, and Evolution" [Seminar(2004)], which was held on the premises of the FORUM2004 [FORUM(2004)] in Barcelona. The automatic cameraman has also been used to record and broadcast real lectures in the amphitheatre at INRIA Rhône-Alpes (example images can be seen in Fig. 6).

3.4 Experimental results

Table 2 shows the confusion matrix for two hours of recording. A ground truth dataset has been produced by manually annotating the situations on the video. The three situations are :

- *Slides :* a new slide is projected on the screen

- *Speaker :* the speaker is talking or answering a question
- *Audience :* someone in the audience is asking a question

The lines of the matrix contain the detection results, while the columns contain the expected response.

	Slides	Speaker	Audience
Slides	0.96	0.04	0.0
Speaker	0.0	0.89	0.11
Audience	0.37	0.05	0.58

Table 2: Confusion matrix

We have obtained a recognition rate of 93.7%. As the context determination is deterministic, this confusion matrix mainly shows the robustness of the underlying perception algorithms (the speech activity and the "new slide" detector). In the last line of the matrix, we can see that there is a big confusion between "audience asking a question" and "new slide". Indeed, when someone starts answering a question, at the same time, he is often seeking for slides in his presentation. This example illustrates that in some case, deterministic approach can be advantageously replaced by more fuzzy ones.

We have also made an informal survey after broadcasting the lectures in the amphitheatre at INRIA. Remote spectators could access two streams : a first stream provided by a fix camera and a stream provided by our automatic cameraman. They all prefered the automatic cameraman stream, allowing them to better understand the presentation.

4 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 [Vidal(2005)a]. A probabilistic finite-state machine is a probabilistic automaton (PFA) defined over a finite alphabet $\sum$. A language is a subset of $\sum^*$. A PFA defines a stochastic language, which is a probability distribution over $\sum^*$. The distribution must verify:

$$\sum_{x \in \sum^*} Probability(x) = 1$$

A formal definition of PFA can be found in [Vidal(2005)a].

4.1 Hidden Markov Models

A hidden Markov model (HMM) [Rabiner(1989)] 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. A more formal definition of an HMM is given in [Rabiner(1989)]. The two following propositions hold [Vidal(2005)b]:

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.

4.2 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, roles, or relations) 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 on 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 events).

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

[Rabiner(1989)] 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).

4.3 Example : Detection of Interaction Groups

This example addresses the problem of detecting changing interaction group configurations in a smart environment. During a meeting, participants can form one big group working on the same task, or they can split into subgroups doing independent tasks in parallel. Our objective is to determine the current small group configuration, i.e. who is interacting with whom and thus which interaction groups are formed. As we focus on verbal interaction, one group has a minimum size of two individuals (assuming that isolated individuals do not speak). The speech of each meeting participant is recorded using a lapel microphone. An automatic speech detector [FAME(2004)] parses this multi-channel audio input and detects which participant stops and starts speaking. We admit the use of lapel microphones in order to minimize correlation errors of speech activity of different participants, i.e. speech of participant A is detected as speech of participant B.

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 the automatic speech detector. Fig. 10 shows the situation network for a meeting with 4 participants. Each possible interaction group configuration is represented by one situation. These situations are transformed to the states of a HMM (Fig. 11).

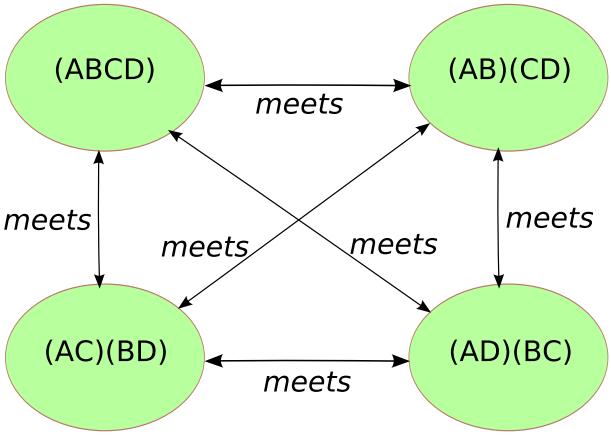


Figure 10: Situation model for a meeting of 4 participants A, B, C, D.

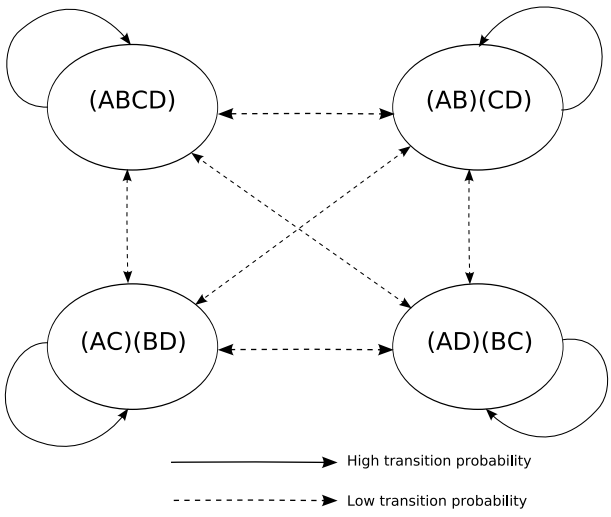


Figure 11: States of the HMM implementation of the situation model for a meeting of 4 participants A, B, C, D.

The probability distributions of the different states are specified based on conversational hypotheses. These conversational hypotheses assume that speech within an interaction group is more regulated than speech between distinct interaction groups. 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 4.2) to the flow of arriving observations.

Figure 12: Example of a configuration of 2 groups of 2 participants

4.4 Experimental results

To evaluate, we recorded three small group meetings with 4 participants (Fig. 12). Using the HMM detector, we obtained a total recognition rate for the small group configurations of 84.8 % [Brdiczka(2005)]. Table 3 shows the confusion matrix for the 3 experiments. This matrix indicates for each group configuration the correct and wrong detections. The lines of the matrix contain the detection results, while the columns contain the expected response.

The figure 13 gives an overview of the detection of group configurations over time. The lines of the chart correspond to different group configurations. The continuous line indicates the correct group configuration expected as detection result.

The results are encouraging and tend to validate the conversational hypotheses to distinguish

	(ABCD)	(AB)(CD)	(AC)(BD)	(AD)(CB)
(ABCD)	0.88	0.03	0.06	0.03
(AB)(CD)	0.08	0.87	0.05	0.00
(AC)(BD)	0.22	0.01	0.77	0.00
(AD)(CB)	0.04	0.04	0.08	0.84

Table 3: Confusion matrix

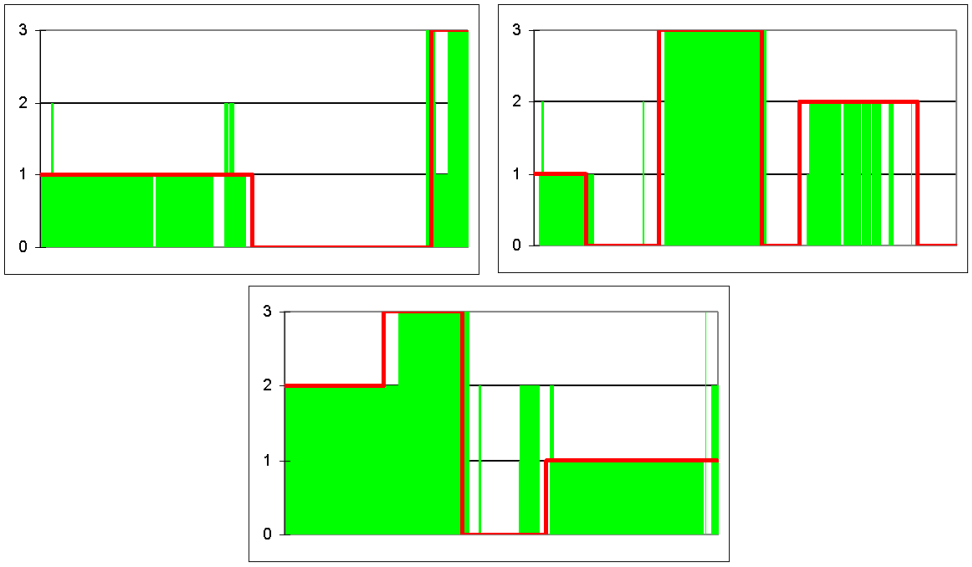


Figure 13: The upper left figure corresponds to the group configuration detection for the experiment 1 (duration: 9 min. 22 sec.). The upper right figure corresponds to the experiment 2 (duration: 15 min. 16 sec.). The third figure corresponds to the experiment 3 (duration: 16min. 19sec.) 0 on the axis corresponds to the group (ABCD), 1 is for the two groups (AB) and (CD), 2 for (AC)(BC) and 3 for (AD)(BC).

interaction groups. The Viterbi algorithm executed on long observation sequences is quite robust to wrong detections of the speech activity detector. However, a minimum number of correct speech activity detections is necessary, as the method relies on the information of who speaks at which moment. The use of lapel microphones made it possible to limit wrong detections as these microphones are attached to a particular person (and thus should only detect his/her speech).

5 Conclusion

This article proposed two different implementations for the situation model representing context: a deterministic one based on Petri nets and a probabilistic one based on hidden Markov models. Both approaches have been applied to real world problems with success: an automatic cameraman system (Petri nets) and an interaction group detector (HMMs) have been implemented.

Each implementation is well adapted for particular applications. Petri nets can implement all Allen temporal operators, in particular those describing parallelism. However, Petri nets can not model erroneous perceptions and uncertain situations (uncertain expectations of perceptions for a situation and uncertain situation transitions). Hidden Markov models are less rich in modelling temporal constraints (in particular parallelism), but HMMs permit to model erroneous input and uncertain situations. Thus the choice of the implementation depends on the application that is envisaged.

References

[Allen(1983)] J., Maintaining Knowledge about Temporal Intervals, Communications of the ACM, 26 (11):832-843, 1983.

[Brdiczka(2005)] Brdiczka, O., Maisonnasse, J., and Reignier, P., Automatic detection of interaction groups. Proceedings of the International Conference on Multimodal Interfaces, Trento, Italy, October 2005.

[Crowley(2002)] Crowley, J.L., Coutaz, J., Rey, G. and Reignier, P., Perceptual Components for Context Aware Computing, Proceedings of the International Conference on Ubiquitous Computing, Göteborg 2002.

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

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

[FAME(2004)] FAME: Facilitating Agent for Multi-Cultural Exchange (WP4), European Commission project IST- 2000-28323 October 2001.

[FORUM(2004)] FORUM2004: Universal Forum of Cultures, Barcelona, July 2004, http://www.barcelona2004.org.

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

[Loke(2005)] 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.

[Muehlenbrock(2004)] 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.

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

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

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

[Seminar(2004)] Consorci Universitat Internacional Menéndez Pelayo de Barcelona, "Tecnologies de la llengua: darrers avenços" and "Llenguatge, cognició i evolució", Barcelona, July 2004, http://www.cuimpb.es.

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

[Vidal(2005)a] 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.

[Vidal(2005)b] 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.

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