Situation Recognition: Representation and Algorithms

Christophe DOUSSON, Paul GABORIT and Malik GHALLAB
dousson@laas.fr, gaborir@laas.fr, malik@laas.fr
LAAS-CNRS, 7 ave du Colonel Roche
31077 TOULOUSE Cedex, FRANCE

Abstract

The situation recognition system, to which this paper is devoted, receives as input a stream of time-stamped events; it performs recognition of instances of occurring situations, as they are developing, and it generates as output deduced events and actions to trigger. It is mainly a temporal reasoning system. It is predictive in the sense that it predicts forthcoming events relevant to its task, it focuses its attention on them and it maintains their temporal windows of relevance. Its main functionality is to recognize efficiently complex temporal patterns on the fly, while they are taking place. This system has been tested for the surveillance of an environment by a multisensory perception machine; it is being applied to monitoring a complex dynamic system.

1 Introduction

We are interested in situations that arc not static states, but do correspond to evolutions of a changing environment. We understand a situation assessment system as one which has to maintain, through perception, a coherent interpretation of what is going on in a dynamic world. Such a task arises in applications like environment surveillance or process monitoring.

Our work was initially motivated by the former class of applications. More specifically, we developed it along with the design of an active multi-sensory perception machine, called SKIDS, that has mobile and fixed cameras, laser range finders, optical barriers, and sonic detectors distributed over and surveying an indoor environment [GRANDJEAN91, GHALLAB92]. A sensory stimulus, such as an optical barrier crossing, or a track detection, once interpreted, becomes an event. A set of events, occurring in some temporal pattern, may develop into a situation (complete or partial), which in turn can generate new events, can permit the focus of attention enabling the detection of forthcoming events, or can trigger alarms, messages, data logging or also other actions.

The proposed situation recognition system has been tested recently for monitoring dynamic systems. Observed events are generated from raw data through simple signal processing. Sucessfull results obtained on a reduced example (a regulated tank) lead us to start a more ambitious application for the surveillance of a gas turbine.

Initially, the programmer provides the system with a set of situation models, or scenarii, of normal and abnormal evolutions to be surveyed. Each situation model is a set of event patterns and temporal constraints between them and with respect to the context. If some observed events match the event patterns, and if their times of occurrence meet the specified constraints, then an instance of this situation occurs. A situation model may also specify events to be generated and actions to be triggered as a result of the situation occurrence. Deduced events can in turn be taken as input by other situations, hence enabling a recursive chaining.

The situation recognition system receives as input a stream of time-stamped events, not necessarily sorted according to their occurrence dates (there may be variable delays for the interpretation of sensory stimulus). It performs recognition of instances of occurring situations, as they arc developing and it generates as output deduced events and triggered actions. It is mainly a temporal reasoning system. It is predictive in the sense that it predicts forthcoming events that are relevant to partial instances of situations currently taking place; it focuses on them and it maintains their temporal windows of relevance. However, as it is today, it does not perform neither temporal projection [MCDERMOTT82, DEAN87], nor persistence maintenance [DEAN83, MATERNE91], through domain axioms and models of change. Its main functionality is to be able to recognize efficiently complex temporal patterns on the fly, while they are taking place.

The AI literature reports on several works with concerns similar to ours. There is the plan recognition problem [WILENSKY 8. KAUTZ86], where one is interested in recognizing that a sequence of actions makes some complex plan. However in such a problem temporal reasoning docs not arise as the main issue. There is also the event calculus [KOWALSKI86, BORILLO90], or some variant of it [BORCHARDT85], which is relevant to our work. But this calculus is mainly interested in describing relations between events to enable a question-answering system to relate them; it does not address the recognition issue.

More akin to our approach is the work of [KUMAR87]. There, a state-based model of time is taken. An extension of

the interval calculus is proposed to relate intervals for which only initial points are known. Events are intervals; they are linked in this extended calculus to form situations which are recognized by some evaluation process; no algorithm is given. The main differences with the approach proposed here are:

- at the knowledge representation level: ours is richer and more realistic, it takes into account the context and distinguishes between occurrence and reception dates of events, it also permits numerical temporal constraints;

- and mainly at the recognition level and at the algorithms involved in it, since we are proposing efficient on-line processes suitable for demanding applications.

Section II describes the proposed knowledge and temporal representation and defines situations as models of temporally constrained events and accompanying actions. Then we will develop in section III the recognition method and main algorithms. We will conclude with a presentation of the experiments ran on the actual implementation and an experimental model of the average complexity that exhibits good performances.

2 Representation

2.1 Time

For algorithmic complexity reasons our time-map manager, called IXTET [GHALLAB89], relies on time-points as elementary primitives. We consider time as a linearly ordered discrete set of instants, whose resolution is sufficient for the dynamics of the environment (i.e., any change can be adequately represented as taking place at some instant of the set). Intervals and relations of the restricted interval algebra [VILAIN86, ALLEN83] can also be represented at the user level in IXTET, but they are translated internally into time-point constraints (we'll not develop that issue here). We can handle the usual symbolic constraints of the time-point algebra (i.e. before, simultaneous, after and their disjunctions), as well as numerical constraints. The later are expressed as pairs of real numbers $I(e_1 \rightarrow e_2) = [I^-, I^+]$ corresponding to lower and upper bounds on the temporal distance between two points e1 and e2.

We use the following notations:

$-I$	=	$[-I^+, -I^-]$	
$I + J$	=	$[I^- + J^-, I^+ + J^+]$	(propagation)
$I \cap J$	=	$[\max(I^-, J^-), \min(I^+, J^+)]$	(conjunction)
$d + J$	=	$[d + J^-, d + J^+]$	(translation)

For complexity reasons, we do not allow disjunctions of numerical constraints, since the constraint satisfaction problem becomes NP-Complete [DECHTER91 J.

2.2 Assertions and events

We rely on a propositional reified logic formalism [SHOHAM87], where a set D of propositions is temporally qualified by predicates such as *true, false, on* and *off* At each time point t and for each proposition P of D, we have either TRUE(P,t) or FALSE(P,t). We consider assertions on the truth of a proposition P over some period of time:

$$\text{TRUE}(P, t, t') \equiv \forall \tau \in [t, t'[, \text{TRUE}(P, \tau)$$
$$\text{FALSE}(P, t, t') \equiv \forall \tau \in [t, t'[, \text{FALSE}(P, \tau)$$

We define an event pattern e as a change in the truth value of a proposition. An event e is a time stamped instance of pattern e, it has no duration, it is expressed by the predicates *ON or OFF*

$$\text{ON}(P,t) \equiv \exists \tau < t < \tau' \mid \text{FALSE}(P, \tau, t) \wedge \text{TRUE}(P, t, \tau')$$
$$\text{OFF}(P,t) \equiv \exists \tau < t < \tau' \mid \text{TRUE}(P, \tau, t) \wedge \text{FALSE}(P, t, \tau')$$

We suppose that the event stream is consistent, i.e., for each proposition, there is necessarily an *OFF* event between two successive *ON* events (and respectively for *ON).* This assumption is solely with respect to the occurrence dates of events, it does not concern the stream as received by the system.

2.3 Processing delays of events

An event e of pattern ε comes with a variable delay, due to sensor processing and data transmission. We suppose that this delay is bounded by an interval $\Delta(\varepsilon)$ given by the user. If d(e) is the occurrence date of event e and r(e) its reception date, we always have: $r(e) \in \Delta(\varepsilon) + d(e)$.

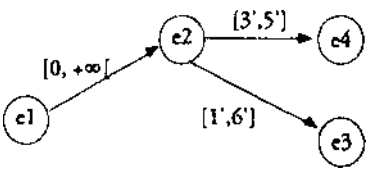
To simplify the notation in following sections, we may refer to e both, as the name of the event, and as its occurrence date d(e).

2.4 Situations

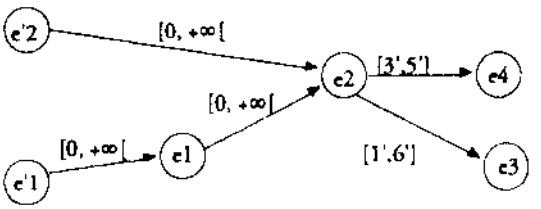
A situation model Σ is a set of event patterns and a set of constraints. It may involve also the description of some context that is required to hold independently of when it became true. For example, we want to survey the work of a robot which must enter a room, load a machine and then exit. As soon as the machine is loaded, it can start its work. The machine must be stopped before loading it and a safety condition is necessary for all actions of the robot in the room. This situation model is described as follows:

```
situation RobotLoadMachine {
    ON(RobotInRoom,e1);
    OFF(RobotInRoom,e4);
    ON(MachineLoaded,e2);
    ON(MachineRunning,e3);
    e1 < e2;
    1' ≤ e3 - e2 ≤ 6';
    3' ≤ e4 - e2 ≤ 5';
    FALSE(MachineRunning,e2);
    TRUE(SafetyCondition,e1,e4);
    when recognized
            emit ON(SucessfullLoad, e2);
}
```

A network representing this situation is the following:



The system receives and processes only events. Since we assume a consistent stream of events, assertions are managed quite naturally through occurrences and non-occurrences of events. To process the assertion TRUE(P,t_1,t_2), the system checks that there has been an event ON(P,t) with $t<t_1$ and such that no event OFF(P,t') occurs with t' in $[t,t_2[$. To initialize the system, a set of events corresponding to the state of the world must be given with an occurrence date equal of $-\infty$, So, the previous network becomes the following.



(e'_1 and e'_2 defined by ON(SafetyCondition, e'_1) and OFF(MachineRunning, e'_2))

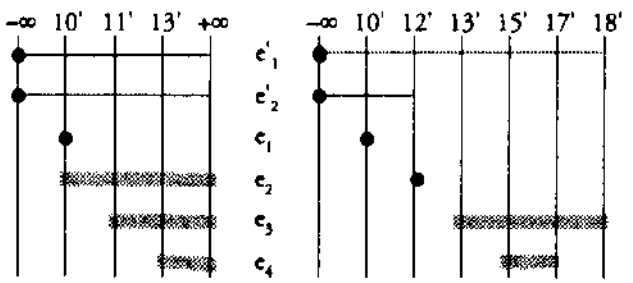
The user can also specify some actions to be performed while a situation instance is being recognized. We may generate a new event with an occurrence date defined relatively to those of the instance events. We may display messages, or run a specified user procedure (external action like in our example). This is a very useful for a *focus of attention* functionnality of the surveillance system. The generation of a new event allows to relate different situations, enabling modular programming.

3 Processing

The situation recognition system must detect on the fly any subset $\{e_i\}$ of the events stream that matches the set of event patterns $\{\varepsilon_i\}$ of a situation model Σ_k (according to all constraints and assertions). A match of some events e_i with a subset of $\{e_i\}$ is a partial instance S_k of the situation model Σ_k. When a complete match is found, the instance is recognized.

3.1 Example

Let us illustrate this process on a simple scenario with respect to the situation model given above: initially SafetyCondition is true and MachineRunning is false. The robot enters the room at time 10' (e_1); loads the machine at time 12' (e_2) which starts running at time 13' (e_3); the robot leaves at time 16' (e_4). On receiving event e_1, the system surveys a possible instance of this situation model, expecting events e_2, e_3 and e_4



in their respecting windows (left figure). Reception of event e_2 is propagated to the windows constraining the remaining events (right figure). Recognition of this instance proceeds accordingly.

If, on some alternate scenario, after receiving e_2, e_4 does not occur at time 17', or if SafetyCondition becomes false before occurrence time of e_4, the corresponding instance will not meet the specified constraints and will be killed.

3.2 Preprocessing

A compilation stage is useful for testing the consistency of constraints in a situation model and for coding it into efficient data structures for the recognition process. This mainly consists in propagating constraints in the event network. At this time, propagations are local to each situation model.

For each situation model, we propagate constraints with a "path consistency" algorithm derived from [MACKWORTH, 85]. The result of this algorithm is (for each situation model) the least constrained complete graph equivalent to the user constraints. To provide a useful feedback for situation debugging, we use an incremental algorithm which can detect inconsistent constraints.

3.3 Recognition delay and ending events.

Ending events are the latest possible events of a situation according to their occurrence dates (i.e. leaf nodes in the corresponding network). *Terminating events* of a situation model Σ are events which can be received the latest by an instance S of Σ before its recognition is completed. To determine these events, the compiler takes into account reception dates and possible delays.

e_i is a terminating event iff

$$\forall\, e_j \in S,\ (I(e_1 \rightarrow e_2)+\Delta(\varepsilon_2)-\Delta(\varepsilon_1)) \cap\ ]-\infty, 0] = \emptyset$$

Proof:

$$d(e_j) - d(e_i) \in I(e_i \rightarrow e_j)$$
$$r(e_j) - d(e_j) \in \Delta(\varepsilon_j)$$
$$d(e_i) - r(e_i) \in -\Delta(\varepsilon_i)$$

thus:

$$r(e_j) - r(e_i) \in I(e_i \rightarrow e_j) + \Delta(\varepsilon_j) - \Delta(\varepsilon_i)$$

e_j is *necessarily* received before e_i iff $r(e_j)-r(e_i) \leq 0$.

Q.E.D.

Due to event delays $\Delta(\varepsilon)$, the recognition of a situation instance can be delayed from its occurrence. It is useful to bound this recognition delay (e_i is a terminating event and e_j an ending event) $\Delta(reco) = [\Delta(reco)^-, \Delta(reco)^+]$ where:

$$\Delta(reco)^- = \min(\Delta(\varepsilon_j) - I(e_j \rightarrow e_i))$$
$$\Delta(reco)^+ = \max(\Delta(\varepsilon_j) - I(e_j \rightarrow e_i))$$

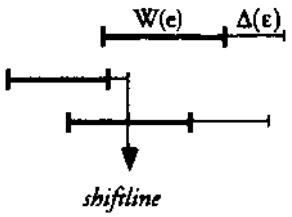
This delay allows to verify that generated events e_k are emitted no later than the user specified bounds $\Delta(\varepsilon_k)$, e.g. "emit e_k at recognition" must verify that $\Delta(reco) \subseteq \Delta(\varepsilon_k)$.

3.4 On line situation management

Our recognition method is based on a complete forecast of forthcoming events predicted by situation models. In order for an event e_i to be consistent with constraints and known dates of other events that already took place in a partial

instance S, we define an interval called window of relevance $W(e_i)$ which contains all the possible occurrence dates of the possibly forthcoming event e_i.

• Time lines: for each situation instance, we process two time lines.



- The shifiline of an instance S is the latest date for the nearest non-occurred event (according to delay Δ).
- The non-occurrence line of an instance is the latest date for ensuring that the nearest assertion holds.

• Recognition forecast we can compute the recognition interval for each situation instance. The user can know when a recognition is expected and which events are required to achieve it.

• Tree of instances: for each situation model, our system manages a tree of current instances. When a situation is recognized or killed (after a constraint violation), the instance is removed from this tree (wc say that the instance dies).

3.5 Evolution of a situation instance

There is basically two ways of modifying a situation instance: a new event can arrive (and can be integrated into an instance or violate a constraint for a protected assertion) or time goes without anything happening and, perhaps, makes some event deadlines to be violated or some assertion constraints obsolete. The system manages the external real time by receiving clock updates. Let now be the value of this clock. So when an event is received, we always have the reception date r(e) = now.

3.5.1 Matching an event occurrence

When event e matching pattern e_k occurs, either :

- $d(e) \notin W(e_k)$: e does not meet temporal constraints on the expected event c_k of S, or
- $d(e) \in W(e_k)$: e meets the constraints, $W(e_k)$ is reduced to [c, e].

More generally, if the window of relevance of some forthcoming event e_k in S has been further constrained (that is $W(e_k)$ reduced), we are sure that this constraint remains consistent with what is already known. We need however to propagate it to other expected events, that in turn are further constrained.

```
propagate(e_k, S)
for all forthcoming event e_i≠e_k of S
    W(e_i) ← W(e_i) ∩ (W(e_k) + I(e_k→e_i))
```

This produces a new set of non-empty and consistent $W(e_i)$. It is important to notice that we never need to verify the consistency of events that fall into their windows of relevance.

In fact, before this propagation, the original situation instance must be duplicated and only the copy is updated by matching e and e_k and by processing propagate(e_k,S).

Duplication is needed to guaranty the recognition of a situation as often as it may arise, even if its instances are temporally overlapping (we will see in 3.4 how to limit the profusion of possible instances). This is illustrated by the following situation model:

e_1 before e_2 before e_3 with $(e_3 - e_2) = 3'$

if we receive the stream of events:

e_1 at time 0', e_2 at 3', e_2 at 8', e_3 at 11'

The first occurrence of e_2 (if matched) will kill the situation at time *now*=6' because of the non occurrence of e_3. By generating a second instance of the situation at *now*=3' (containing only e_2), we can recognize the instance matching e_2 at 8'.

3.5.2 Current time propagation

When the system sets up its internal clock, this new value of *now* can reduce some windows of relevance $W(e_i)$ and, in this case, we need to propagate it.

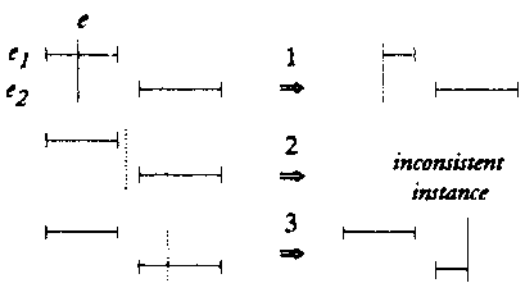
```
propagate-clock(t, S)
now ← t
for all forthcoming events e_i of S
    W(e_i) ← W(e_i) ∩ ([t, +∞[ - D(e_i))
    if W(e_i) = Ø then
        inconsistency, S is killed
    else if W(e_i) was modified then
        propagate(e_i, S)
```

Notice that a clock update does not always require a propagation: it is useless to propagate a date when no timeline has been reached. Indeed, in this case, we are sure that constraints in a situation instance remain consistent. The system computes the minimum value of current timelines; it sents it to the external clock process in order to receive back a clock update only at that moment (if no event occurs meanwhile).

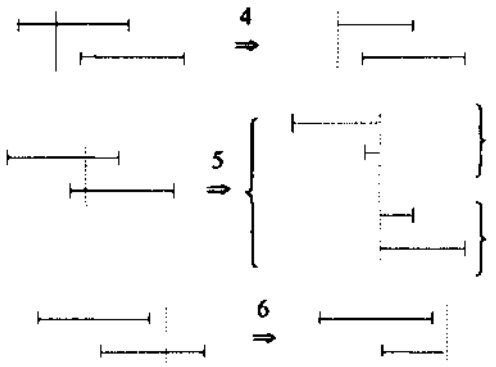
Remark: any situation instance concerned by a new event occurrence must be processed by propagate-clock(now,S) to ensure that each $W(e_i)$ is correctly updated.

3.5.3 Violation of a protected assertion.

Let us consider a situation instance S which requires that an event e must not occur during $[e_1,e_2]$. If e occurs before $W(e_1)$ or after $W(e_2)$, S is not concerned. Otherwise, there are two basic cases (depending on how much $W(e_1)$ and $W(e_2)$ are overlapping) and each one leads to a processing according



1st case: $W(e_1) \cap W(e_2) = \emptyset$



2nd case: $W(e_1) \cap W(e_2) \neq \emptyset$

to the localisation of the time of occurrence of e.
In four cases out of six, we only reduce $W(e_1)$ or $W(e_2)$, but S can also die (case 2) or can be cut into two more restricted instances (case 5). We propagate these new dates by processing **propagate**(e,S) if W(e) is reduced and/or **propagate**(e',S) if W(e') is reduced.

3.6 Restricting duplication of instances

We saw that a situation instance can be duplicated several times during its recognition. This is the main source of complexity and we must limit it. Duration bounds on a situation is the first way to reduce duplication. While compiling, we can verify that every situation has a maximal duration limit. However, even with this limitation, situation instances could be duplicated in large number.s.

Furthermore, there may be situations that cannot have two successful instances overlapping in time or sharing a common event instance; the user can also be interested in recognizing just one instance at a time. Both cases, when a situation instance is recognized, all its pending instances must be removed.

A third way for restricting duplication is to postpone the propagation of some events. There would be two event classes: the completely forecast ones (for which all constraints are propagated) and the partially forecast events. If an event is completely forecast and if d(e) is in W(e), we are sure of the time consistency of e. But a partially forecast event needs more than this simple verification: one has to check through constraint propagation that this event meets the pending constraints. Therefore event matching will be more expensive.

Furthermore, postponing constraint propagation can be expensive: if one does not duplicate a situation when a new event is instantiated, one must keep a history of each possible matching event in order to be able to backtrack when this situation is ended (killed or recognized). We know that backtracking can be very expensive and unacceptable for an on-line system. This is why all events are completely forecast in our current system and there is no constraint postponing.

3.7 Variables in event patterns and situation models

The representation allowed by our system is actually more general than what has been described up to now. It is possible to specify event patterns with variables. Event instances are ground terms. The matching procedure is very simple since the duplication of situation instances enables a direct checking of the consistency of variable bindings.

However our representation is not a full first order formalism, since we do not allow free variables in protected assertions. Indeed constraints on such assertions cannot be processed by simple propagation and forecasting; they require a costly search, not compatible with the complexity we aimed at.

3.8 Complexity

Each elementary event propagation or clock propagation runs in $0(m)$, where m is the number of pending events in a situation instance; $m < n$ the total number of distinct events in the situation model.

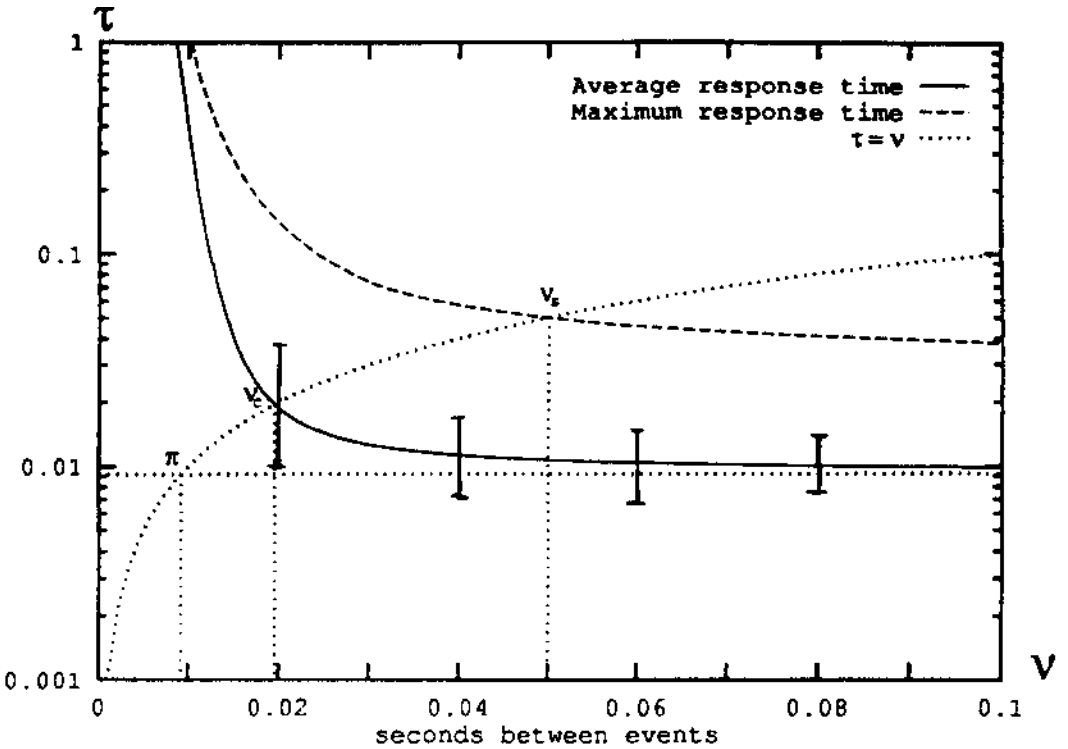
For K instances of situation models with n events each, the proposed algorithms process a new event (or a clock updating) with a complexity in $0(Kn^2)$. K is larger than M, the number of situation models. If K is kept of the same order than M, the overall complexity is quite manageable.

4 Results and Conclusion

The first scenario on which this recognition process was tested describes the surveillance of a mobile robot, bringing and taking objects that are loaded and unloaded by a human operator. The scene takes place into a laboratory environment with 4 rooms separated by 7 optical barriers and surveyed by four cameras. Generated events are optical barriers signals (14 event types) and motion detected by the cameras (8 event types for the robot and 4 for the operator). This scenario was described by 15 situation models, hierarchically organised. The normal cycle is surveyed: if an incidence occurs, the system detects when and where it happens. Live demonstrations of this scenario and others similar were run. Compared to the others reasoning system needed in the SKIDS perception machine, the situation recognition system is fast enough and is not a limiting factor for reactivity: the average response time for an event is of the order of 15ms.

Since these demonstrations, a large number of other experiments were run on different sets of situations (up to one hundred situation models). Our goal was to characterize the practical "bandwidth" of the proposed algorithms and implementation: up to which frequency of events the system is able to follow-up without accumulating delays.

The following figure summarizes the system performances for a set of 80 situation models, each involving about 10 event patterns and as many constraints, 2 or 3 of which referring to the context. Those were abstract but fairly complex situation models, programmed for the purpose of testing different schemes of temporal constraint lattices, and having a significant overlapping in terms of common event patterns (a



total of 400 event patterns were used). We generated 100 scenarios, each corresponding to a sequence of 3000 events meaningfull with respect to temporal constraints of situation models. For each scenario, we ran the recognition system and recorded the processing time for each event. From that, we computed the corresponding response time to an event, as a function of a variable frequency of events, taking into account cumulated delays from one event to the next and the time lost in waiting for an event.

The figure plots the response time τ with respect to ν, the average time between 2 events ($1/\nu$ is the event frequency). π is the average processing time of an event, it does not depend on ν. The figure shows the maximum values and the average values of τ (with some standard deviations). Three interesting points are defined at the intersection of the diagonal $\tau = \nu$ with three curves: maximum response time (ν_s), average response time (ν_c) and average event processing time (π). This shows four domains:

- $\nu_s \leq \nu$: in this secure band there is never a delay
- $\nu_c \leq \nu \leq \nu_s$: there may be a delay from one event to the next, but this remains a safe band since the average response time is smaller than the average period
- $\pi \leq \nu \leq \nu_c$: this is a critical band in which the system may or may not catch up after some sequence of events
- $\nu \leq \pi$: the cumulated delay grows with the number of events; the system cannot follow up in this forbidden band.

Similar curves were obtained for other sets of situation models. The processing time π grows linearly with the number of situations, with values $\pi = 6$, 8, and 9 ms for 20, 50 and 80 models respectively. A similar remark holds for ν_c ($\nu_c = 16$ and 20 ms for 20 and 80 models).

These laboratory experiments show that the practical complexity of the proposed algorithms ensures a quite manageable situation recognition system, even for a highly dynamic environment. We arc working toward a confirmation of these results on more complex industrial applications for alarm filtering in an oil plant and for process monitoring of a gas turbine.

In conclusion, this paper describes an innovative situation recognition system, based on temporal constraint propagation. The proposed representation has been chosen such as to keep the complexity reasonable for on-line processing, while maintaining a sufficient expressiveness for practical applications. The algorithms rely on extended pre-processing and carefully designed data structures, they exhibit good performances, compatible with demanding applications. Overall, the proposed system provides interesting functionalities such as deduction of events on the basis of temporal evolutions recognized on the fly, forecast of expected events and focus of attention.

Our system has been tested in live experiments of the SKIDS perception machine. To our knowledge it is the first time that a perception system integrates explicitly temporal reasonning.

Our next goal is to extend the proposed situation recognition system into a situation assessment system. For that several additional functionalities are required:

- Persistence maintenance of database through domain axioms and models of change to allow a complete deduction of non-observable events; notice that this is partially handled by our system through the possible generation of an event when a situation is recognized;
- Sensory data interpretation is necessarily hypothetical, events and situations should be as well. A situation assessment system must be able to backtrack and manage uncertainty, as well as hypothetical reasoning.

Furthermore we plan to fully investigate the terms of the trade-off beteen event forecasting and propagation postponing, and solve it dynamically and opportunistically.

Acknowledgments

This work benefited from the support of the EEC under ESPRIT projects SKIDS and TIGER.

Christophe DOUSSON is being supported by a scholarship from Shell Research.

The authors are gratefull to their co-workers who helped this work, specially A. Mounir-Alaoui who implemented IXTET kernel [MOUNIR90] and the first version of the situation recognition system.

References

[ALLEN83] ALLEN, J. F. Maintaining Knowledge about Temporal Intervals, Communications of ACM, 26(11):832—843, November 1983.

[BORCHARDT85] BORCHARDT, G. Event Calculus, Proc. in the 9th IJCAI 85.

IBORILLO90] BORILLO, M. and GAUME B. An extension of Kowalski and Sergot's event calculus, Proceedings ECAI, pp. 99-104, 1990.

[DEAN83] DEAN, T. Time Map Maintenance, Yale University, Computer Science Department, 1983.

[DEAN87] DEAN T. and BODDY M. Incremental causal reasoning, ProcAAAI,p. 196-201, 1987.

[DECHTER91] DECHTER, R., MEIRI, I. and PEARL, J. Temporal Constraint networks. Artificial Intelligence 49, p. 61-95, 1991.

[GHALLAB89] GHALLAB M., MOUNIR-ALAOUI A. Managing Efficiently Temporal Relations through Indexed Spanning Trees. Proc. 11th IJCAI, 1297-1303, Detroit, 1989

[GHALLAB92] GHALLAB M., GRANDJEAN P., LACROIX S., THIBAULT J.P. Representations et raisonnement pour une machine de perception multi-sensorielle. p. 121-167, PRC-GDR / IA, Marseille (France), Teknea editions, 1992.

[GRANDJEAN91] GRANDJEAN P., GHALLAB M., DEKNEUVEL E. Multiscnsory scene interpretation: model-based object recognition, p. 1588-1595, IEEE Robotics and Automation, 1991

[KAUTZ86] KAUTZ, H. Generalized plan recognition. Proc AAAI, 1986.

[KOWALSKI86] KOWALSKI R. and SERGOT M. A logic-based calculus of events, New Generation Computing, vol.4, 67-95,1986.

[KUMAR87] KUMAR, K. and MUKERJEE, A. Temporal event conceptualization, Proc. 19th IJCAI, 472-475, 1987.

[MACKWORTH85] MACKWORTH, A.K. and FREUDER, E.C. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. Artificial Intelligence 25, p. 65-74, 1985.

[MATERNE91] MATERNE, S.; HERTZBERG, J. and VOb, H. On clipping persistence (or whatever must be clipped) in Time Maps. Arbeitspapiere der GMD, Sankt Augustin, June 1991.

[MCDERMOTT82] McDERMOTT, D.V. A Temporal Logic for Reasoning about Processes and Plans, Cognitive Science, volume 6, 101-155, 1982.

[MOUNIR90] MOUNIR-ALAOUI, A. Raisonnement temporel pour la planification et la reconnaissance de situations, These de l'UPS, Toulouse, 1990.

[SHOHAM87] SHOHAM, Y. Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence. MIT Press, 1987.

[VILAIN86] VILAIN, M. and KAUTZ, H. Constraint Propagation Algorithms for Temporal Reasoning, Proc. AAAI-86, Philadelphia, Pa., August 1986.

[WILENSKY83] WILENSKY, R. Planning and Understanding, Addison-Weslet Publications, Advanced Book Program, Reading, Mass. 1983