Extending and Unifying Chronicle Representation with Event Counters

Christophe Dousson
France Telecom R&D – 2, avenue Pierre Marzin – F-22307 Lannion Cedex, France
christophe.dousson@rd.francetelecom.com

Abstract.

This paper is dedicated to the chronicle recognition approach used to design an evolution monitoring system for supervising dynamic systems for which time information is relevant.

We propose to extend and also to unify the chronicle representation through event counters. The main motivation of such an extension of the chronicle representation arises from alarm processing: counting the occurrences of alarms can be useful to evaluate the severity of a problem and also to discriminate some kind of faults.

The paper describes the representation and the corresponding algorithms for processing this extension according to the purpose of supervision in terms of performance.

1 Introduction

In this paper, we understand an evolution monitoring system as one that has to maintain through its observations a consistent interpretation of what is going on and the chronicle model representation is used for the recognition of relevant pieces of evolution of the monitored system.

While expert systems base their reasoning on rules, relegating time information to the background, recognition of chronicles is based on diagrams of evolution in which time is fundamental.

Chronicles are temporal patterns that represent *possible* evolutions of the observed system. A chronicle is a set of *events*, linked together by time constraints. In the monitoring framework, these events could be alarms referring to the supervised system. The available time information would allow their ordering and the specification of time spans between two occurrences.

The approach used here is a chronicle recognition approach very similar to [1]: the chronicle recognition system receives as input a stream of time-stamped events. It performs recognition of chronicles as they develop, and generates as output the recognized chronicles (i.e. sets of matched input events). It is mainly a temporal reasoning system based on the complete prediction of the possible arrival dates for each expected event; the set of all these values (called *time window*) is reduced by propagation of the dates of observed events through the graph of the time constraints of the chronicle model. The recognition process is incremental – each event is integrated as soon as it occurs – and it is performed over a single reading of the input stream. This method has a high-performance algorithm, partly due to a stage of chronicle compilation which propagates the time constraints and checks the consistency of the resulting time graph [2].

But, as we will see, this chronicle formalism is not well-suited to take into account transient phenomena that become relevant only when they occur frequently and, moreover, when some faults could only be discriminated by counting alarms. So, we propose to introduce counters of events as an extension of the chronicle formalism.

The next section gives an overview of the chronicle representation and a snapshot of the recognition process. The third section defines the representation of counters of events and the new predicate used in chronicle models. The fourth section details the corresponding algorithms for processing this extension according to the purpose of supervision. Then, just before conclusion, we will propose first results about comparison between the old and the new chronicle recognition system.

2 Recognition of Chronicles

2.1 Representation

This framework relies on a propositional reified logic formalism, where a set of attributes is temporally qualified by predicates [3]. The time representation relies on the time points algebra and we consider time as a linearly ordered *discrete set* of instants whose resolution is sufficient for the environment dynamics.

We suppose that events which are observed by the chronicle recognizer are instantaneous[1] and a chronicle is a description of a time relation between these events.

For instance, let us be the following phenomena: an event a precedes an event c within 2 to 5 units of time, and then, a b and another a occur less than 10 time units after the beginning and we also have an additional constraint $[1, 6]$ between c and b (see fig. 1). To represent this, the chronicle model will be:

$$\begin{array}{lll} event(a, t_1) & \wedge\, event(c, t_2) & \wedge\, (t_2 - t_1 \in [2, 5]) \\ \wedge\, event(b, t_3) & \wedge\, (t_3 - t_1 \in [0, 10]) & \wedge\, (t_3 - t_2 \in [1, 6]) \\ \wedge\, event(a, t_4) & \wedge\, (t_4 - t_1 \in [0, 10]) & \wedge\, (t_2 \leq t_4) \end{array}$$

A predicate $noevent(a, (t_1, t_2))$ is also defined. It means that there is no occurrence of the event pattern a within $[t_1, t_2[$.

2.2 Recognition Process

The chronicle recognition algorithms process the stream of input events in one shot and on-line. Let be a chronicle and an event input stream, the recognition algorithm identifies *all the observed event sets* (called *chronicle instances*) that match the chronicle event patterns in respect with the time constraints. For instance, let's suppose that the supervisor receives the following time-stamped events:

[1] If one must supervise a system where alarms have a duration, we use two instantaneous events which are the beginning and the end of the alarm.

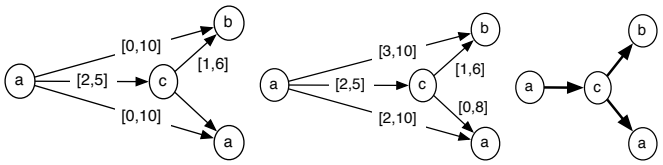


Figure 1. Example of a chronicle. *On the left: the user-defined chronicle. In the center: the chronicle after the constraint propagation. On the right: the (induced) partial order between the events of the chronicle.*

$e_1 : a, t = 4$ $\quad e_2 : d, t = 5$ $\quad e_3 : a, t = 6$ $\quad e_4 : c, t = 8$
$e_5 : b, t = 10$ $\quad e_6 : e, t = 11$ $\quad e_7 : a, t = 12$ $\quad e_8 : b, t = 14$

The chronicle given as an example in the figure 1 is recognized three times: the matched instances are $\{e_1, e_4, e_5, e_7\}$, $\{e_3, e_4, e_5, e_7\}$, and $\{e_3, e_4, e_7, e_8\}$. In general, events may be shared by many chronicles and the system is able to manage all the concurrent instances. It should be emphasized that the recognition is exhaustive, i.e. all the possible instances of the defined chronicles are identified by the system.

To achieve these recognitions, the process manages a set of partial instances of chronicle as a set of time windows (one for each forthcoming event) that is gradually constrained by each new matched event: this system is predictive in the sense that it predicts forthcoming events that are relevant to instances of chronicles currently under development; it focuses on them and it maintains their time windows (more details can be found in [2]).

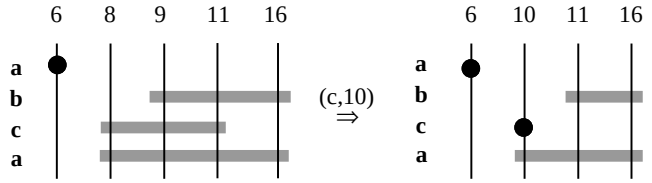


Figure 2. Partial instance evolution. *The time windows of a partial instance* $\{(a, 6)\}$ *(left hand) and the effect of the time constraint propagation due to the integration of* $(c, 10)$ *(right hand).*

2.3 Applications

Chronicle representation has been recently used in the AUSTRAL project in order to analyze a sequence of alarms emitted by substations in a French medium voltage distribution network [4] . It is also used in the GASPAR project in order to analyze alarms issued by equipment in a telecommunication network[5]. It is also used in some subtasks of the WITAS project [6, 7] to represent car behaviors in the supervised road traffic. New applications areas for temporal recognition systems have been proposed in medical domain such as hepatitis symptoms tracking [8], intelligent patient monitoring [9] or cardiac arrhythmia detection [10]. A more complete overview of chronicle recognition applications can be found in [11].

3 Counting Alarms in Chronicles

3.1 Motivation

The main motivation of extending the chronicle representation arises from alarm processing: many occurrences of an alarm can be generated by a persistent problem and counting these occurrences can be useful to evaluate the severity of the problem. Moreover, some faults could only be discriminated by counting alarms.

For instance, in the greatest french packet switching telecommunications network, we know the two following faults:

- F1: when a technical center (which groups many substations) shutdowns and reboots, it sends to the supervision center one "reboot" event and each of its substations sends a "OK status" event,
- F2: when a substation comes down and restarts, it sends also a "OK status" event.

But, in case of many faults occurrences, the "reboot" and some "OK status" events of F1 can be lost and, as a consequence, we can't distinguish a technical center breakdown (F1) from many (independent) substation problems (F2). In a practical way, experts exhibit empirical characteristics to deal with these losses: when we receive less than n "OK status" events during 3 minutes ($3 \leq n \leq 5$ depending of the considered technical center), we probably observe multiple F2 and, over this threshold, we consider F1.

As these losses are due to delays in buffer reinitialization in telecommunications equipment, this ambiguous situation is not rare and we use a counting threshold as often as we encounter a such discrimation problem that involves many pieces of equipment (and so many alarms of the same type are generated).

Now, let us suppose that we would write a chronicle that receiving more than 3 occurrences of event pattern a but less than 5 occurrences within 30 seconds. With the current representation, we must explicit three chronicles, respectively with 3, 4 and 5 alarms, and you also must ensure that there is no more occurrences of a by using the $noevent$ predicate[2]. Here follows one of the needed chronicles (remembering that time is considered as an ordered discrete set and that $noevent$ is defined on $[t_1, t_2[$):

$$\begin{array}{ll} event(a, t_1) \wedge event(a, t_2) & \wedge\, event(a, t_3) \\ \wedge\, noevent(a, (t_0, t_1)) & \wedge\, noevent(a, (t_1{+}1, t_2)) \\ \wedge\, noevent(a, (t_2{+}1, t_3)) & \wedge\, noevent(a, (t_3{+}1, t_4)) \\ \wedge\, (t_0{<}t_1{<}t_2{<}t_3{<}t_4) & \wedge\, (t_4{-}t_0 = 30) \end{array}$$

So, we propose a new predicate that allows to express event counters in order to write easily and also to alleviate the computational part of the recognition of chronicles with counters.

3.2 The Counting Predicate

We introduce a new predicate $occurs$ as follows:

$$occurs((n_1, n_2), a, (t_1, t_2)), with\ 0 \leq n_1 \leq n_2$$

This predicate means that, between the two timepoints t_1 and t_2, there are exactly N occurrences of event that match the pattern a and $n_1 \leq N \leq n_2$. In other words, if we denote $e_i = (a, d_i)$ the occurrences of pattern a at date d_i, we have:

$$n_1 \leq Card\{e_i | t_1 \leq d_i < t_2\} \leq n_2$$

As a natural extension, we allow $+\infty$ instead of n_2 (i.e. that there are more than n_1 occurrences in the associated time interval). Notice that, for consistency with the old $noevent$ predicate definition, we count occurrences on the interval $[t_1, t_2[$.

[2] Instead of growing the number of chronicle models, we can use *optional* events in chronicle models as proposed in [4] but we can't avoid the increase of the number of predicates which is directly connected with the complexity of the recognition algorithms.

This new predicate can actually be proposed as a unification of the chronicle language since the old predicates can be replaced with particular uses of $occurs$. The $noevent$ predicate is obviously:

$$noevent(a,(t_1,t_2)) \equiv occurs((0,0),a,(t_1,t_2))$$

The predicate $event$ is an existantial predicate that means that there is *at least* one event, so it is logically equivalent to[3]:

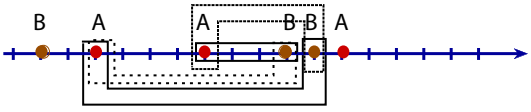
$$event(a,t) \equiv occurs((1,+\infty),a,(t,t+1))$$

Moreover, it is easy to define optional events as proposed in [4] through this predicate by writing:

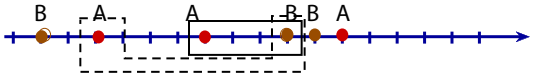
$$occurs((0,1),a,(t,t+1))$$

The figure 3 shows an example of an input stream and the associated output of the recognition process according to different uses of the $occurs$ predicate.

$occurs((1,1),A,(t,t{+}1)) \wedge occurs((1,1),B,(t',t'{+}1)) \wedge (t \leq t')$:



$occurs((1,1),A,(t,t+1)) \wedge occurs((1,1),B,(t,t'))$:



$occurs((1,1),A,(t,t')) \wedge occurs((1,1),B,(t,t'))$:

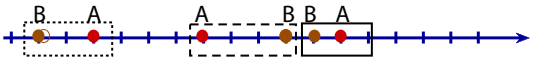


Figure 3. Different uses of predicate $occurs$ and the corresponding recognized chronicles.

4 Algorithms

The chronicle recognition system must detect on the fly any subset $\{e_i\}$ of the event stream that matches the set of event patterns in respect with the time constraints of the chronicle model. When a complete match is found, the instance is recognized. As we use the time propagation algorithm as detailed in [2], we don't focus on that point in this paper.

For each instance C, we use the following notations:

- $\Gamma(C)$ as the set of the predicates that must be asserted (when this set is empty, the instance C is recognized).
- $W(C,t_i)$ as the time window relevant to the instant t_i of the instance C, this interval is always updated during the recognition process in order to contain *all the possible values* for the temporal variable t_i. We use respectively $W(C,t_i)^-$ and $W(C,t_i)^+$ as the lower and the upper bounds.
- $\Omega(C)$ as the set of the matched events of the instance C.

[3] Notice that, in many applications, the input stream is not supposed to have more than one event of each type at the same date and so, $occurs((1,+\infty),a,(t,t+1)) \equiv occurs((1,1),a,(t,t+1))$.

There are two phenomena that the system must take into account: the new occurrence of an event and the increase of the current time (when time passes with nothing occurs). The two following subsections will describe the corresponding algorithms.

4.1 Event Integration

The event integration in a counter predicate is similar to the integration of an event in a protected assertion ($noevent$) as detailed in [2].

We suppose that the system receives an event (a,d) and tries to integrate it in the predicate $occurs(n_1,n_2,a,(t_1,t_2))$ of a chronicle C. We must consider the three following hypotheses:

1. $t_2 \leq d$: the event is too late.
2. $t_1 \leq d < t_2$: the event must be integrated (and included in $\Omega(C)$).
3. $d < t_1$: the event is too early.

As we want to integrate an event as soon as it is received by the system, we duplicate the current instance C into one to three exclusive hypothesis depending on the relative position of the event occurrence date and the instants of the predicate (which are no necessary already valued but only bounded by their time windows).

We have many cases to consider, depending to the relative position of t_1, t_2 and d (we have $t_1 < t_2$, so that $W(C,t_1)^- < W(C,t_2)^-$ and $W(C,t_1)^+ < W(C,t_2)^+$). The three following cases are trivial:

- $d < W(C,t_1)^-$: we are sure that the event is too early, all the possible values for t_1 are strictly after d, this event can't be integrated in C, there is nothing to do.
- $W(C,t_2)^+ \leq d$: we are sure that the event is too late for this predicate, as in the previous case, there is nothing to do.
- $W(C,t_1)^+ \leq d < W(C,t_2)^-$: we are sure that the event occurs between t_1 and t_2 and so it must be integrated in C (and the counter of the predicate is increased).

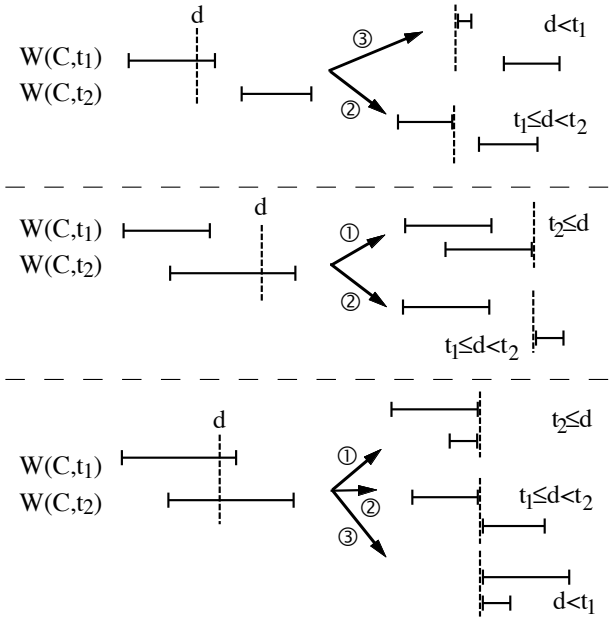


Figure 4. The three cases of duplication of instances when integrating an event (depending on the relative position of the event occurrence date d and the time window of t_1 and t_2. All the hypotheses #2 accept the new event occurrence in the chronicle and increase their associated counters.

And the three other cases which are displayed on figure 4 lead to duplication of the instance C in order to consider all the hypotheses of ordering t_1, t_2 and d: the topmost case can evolve in two hypotheses (#2 and #3); the second case gives hypotheses #1 and #2 and the last one produces all the three hypotheses.

For all the instances, as the time windows of t_1 and/or t_2 are reduced by $[d, +\infty[$ or $]-\infty, d]$, we also need to propagate these new time windows through the time constraints graph of the chronicle.

Moreover, when an event is integrated (hypothesis #2), we increase the predicate counter of the chronicle. If all the predicates of the chronicle become greater than their n_1, the chronicle is called *ready to recognized* but is not recognized at this moment since it is already possible that the maximum bound will be reached by next events. Indeed, if the counter becomes greater that n_2, the instance is killed since one predicate of the chronicle is violated. The recognition could only occurs when time passes and takes all the $W(t_2)^+$ over (see next section).

4.2 Clock Propagation

We must consider two different cases, depending on the kind of the partial instances:

- *Missing events* instance (common case): we compute a timeline that indicates the deadline which has an effect on time windows and so, must be propagated. A predicate can be affected if time windows are reduced in order to respect $W(C, t_2)^+ \leq now$. If it is the case, it can be asserted (if the counter reached n_1) and removed from $\Gamma(C)$ or it can be violated (if it is has not enough events) and the instance is killed.

$$MissLine(C) = \min_{\Gamma(C)}\{W(C, t_i)^+\}$$

- *Ready to recognized* instance (the lower bounds n_1 of *all* the counters are reached): we compute a timeline that gives us the right to recognize the instance when the current clock reaches this *RecoLine* (no counter can be violated after this deadline).

$$RecoLine(C) = \max_{\Gamma(C)}\{W(C, t_i)^-\}$$

4.3 Instances Tree Management

In order to manage efficiently all the partial instances, the chronicle recognition system stores all these hypotheses in trees, one for each chronicle model. Each occurrence of event and each clock tick traverses these trees in order to kill some instances (nodes) or to develop some hypotheses or to process recognitions as shown in previous section.

We integrate a new occurrence of event as follows (see figure 5):

1. the hypothesis $d \geq t_2$ (event too late) is stored as a sister of the current instance.
2. the hypothesis $t_1 \leq d < t_2$ (event integration) is stored as a child of the current instance.
3. the hypothesis $d < t_1$ (event too early) is stored in place of the current instance

We can prove that the trees verify the following property:

Theorem 1 *if an event does not concern an instance C (i.e. it does not match any predicate in $\Gamma(C)$), then it does not concern any children of C.*

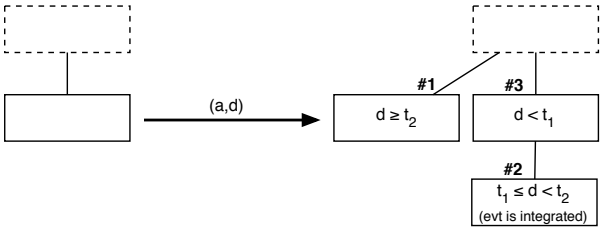


Figure 5. Storing the new hypotheses in the instances tree.

So, we can use this property to jump some branches and traversing the tree can be optimized as it was done in the previous system (see [2]): the process of a chronicle with counters does not affect this point.

5 Performances

The counter representation proposed here is more efficient only because it could express chronicles is a more concise way: the use of counters in the old representation increases the number of chronicles models used by the recognition system.

Let us suppose that we have a chronicle that contains $occurs((n_1, n_2), a, (t_1, t_2)$. With the old representation, we must have $n_2 - n_1 + 1$ different chronicle models (one with n_1 events, one with size $n_1 + 1$, ..., and one with size n_2). Refer to the end of section 3.1 for an example.

As the performance of the recognition system is directly connected with the number of the developed chronicles instances (the number of hypotheses), all our tests and results are comparisons between the number of hypotheses developed by the old and the new system.[4]

5.1 Theoretical Case

This theoretical case is built to exhibit the behavior of counters: let's suppose that the input event stream is a sequence of events a which are regularly time-spaced, with a frequency f (i.e. there is a delay $\Delta = 1/f$ between two consecutive events) and that we want to match the following pattern: $occurs((n_1, n_2), a, (t, t + k.\Delta))$. In other words, we count events during $T = k.\Delta$.

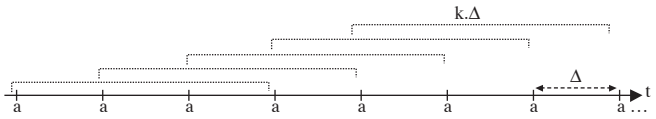


Figure 6. Recognized chronicles with the pattern $occurs((n_1, n_2), a, (t, t + k.\Delta))$ in a regularly time-spaced stream.

With the old representation, we have $n_2 - n_1 + 1$ chronicle models (one with n_1 events, one with size $n_1 + 1$, ..., and one with size n_2). The number of created instances of one of these models depends on the comparison between the size n of the chronicle model and k:

- $k < n$*:* these chronicles live until their first events are too old and, finally, chronicles die because there are not enough events for them; there are always $k - 1$ chronicles like this (one begin at $now - \Delta$, another at $now - 2\Delta$,..., and one at $now - (k-1)\Delta$).

[4] As the old system, the process of *one* instance has a complexity of $O(n)$ where n is the size of the time graph of the chronicle.

When an event a occurs, all of them are duplicated and a new one is created, so k new instances are created.

- $k = n$: as above, except chronicles are not killed but recognized,
- $k > n$: these chronicles are killed because there are too many events in them; there are always $n - 1$ chronicles to duplicate and one to create. When an event a occurs, n new instances are created.

So, when an event a occurs, the system creates $min(k, n)$ new instances for *each* model. To get the number of created instances for each new event occurrence, we sum all the created instances for all the models (figure 7 shows the curve for a counter defined by $n_1 = 3$ and $n_2 = 7$):

$$\sum_{n=n_1}^{n_2} min(k,n) = \begin{cases} (n_2 - n_1 + 1).k & (k \leq n_1) \\ -\frac{1}{2}k^2 + \frac{2n_2+1}{2}.k - \frac{n_1^2 - n_1}{2} & (n_1 \leq k \leq n_2) \\ \frac{n_2(n_2+1)}{2} - \frac{n_1(n_1-1)}{2} & (k \geq n_2) \end{cases}$$

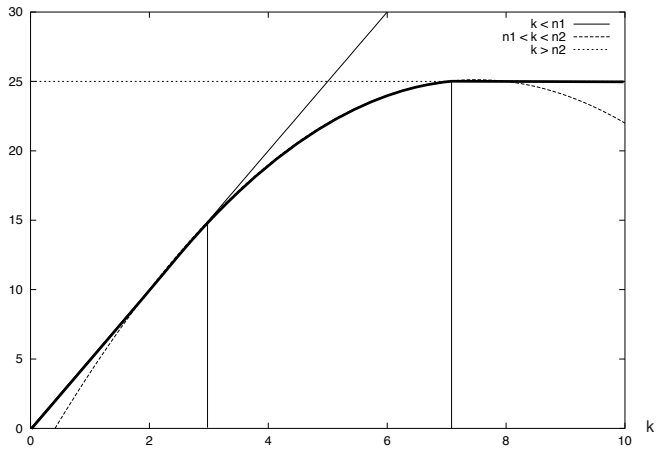


Figure 7. Number of created instances of $occurs((3,7), a, (t, t + k.\Delta))$ by the old system for *each* new event a (the new system creates only one).

On the other hand, the algorithms detailed in section 4 creates only one new instance for each new event occurrence: this is the best case for the new system, there is no useless creation since each partial instance leads to a recognition.

Notice that when we only use the old $event$ predicate, and as we have only one event a at a time in this theoretical case, we could consider $n_1 = n_2 = 1$ and so one instance is created at a time for both systems.

5.2 Empirical Results

We also made empirical measurements of the number of generated hypotheses (partial instances). We randomly generated ten chronicles with event counters with a maximum bound less than six events.

We wrote these chronicles with counting predicates for the new system and the corresponding ones only by using $event$ and $noevent$ predicates, as detailed at the end of section 3.1 for the old one (this corresponds to about forty chronicles). We fed these chronicles with the logs of about 1000 events and count the hypotheses generated by both systems (of course, the *recognized* chronicles are the same since the translation follows a logical equivalence).

The following table shows the number of generated hypotheses by the system with the old system (without $occurs$) and the new one (with $occurs$). Obviously, the number of recognitions are the same. The parameter Δ gives the average delay between two consecutive events in the input stream.

Δ	Reco	old CRS	new CRS	ratio
4.418	165	35033	4712	7.43
7.025	105	32456	4722	6.87
9.64	71	26139	4411	5.93
12.1	58	23056	4348	5.3

Of course, this was a specific (and private) evaluation. Nevertheless, these results show that it could be a significant benefit to explicitly represent counters in chronicles.

6 Conclusion

We present in this paper a useful extension of chronicle representation which are the counters of events. The new predicate does not increase the complexity of the representation but leads to a unification of all the predicates.

We saw that the expressiveness is significantly increased by the introduction of alarm counters without increasing the algorithms complexity. First comparisons between the old and the new formalism show that, when events counters are useful, there is a significant reduction of the number of developed hypothesis during the recognition process. The additional complexity of the extension can be neglected regards to this alleviation and the more condensed representation of the chronicle model makes the system globally more efficient.

REFERENCES

[1] M. Ghallab, "On chronicles : Representation, on-line recognition and learning," *Proc. of the 5^{th} International Conference on Principles of Knowledge Representation and Reasoning (KR-96)*, pp. 597–606, Nov. 1996. Morgan-Kauffman.

[2] C. Dousson, P. Gaborit, and M. Ghallab, "Situation Recognition: Representation and Algorithms.," in *Proc. of the 13^{th} IJCAI*, vol. 1, (Chambéry, France), pp. 166–172, Aug. 1993.

[3] Y. Shoham, "Temporal logics in AI: semantical and ontological considerations," *Journal of artificial intelligence*, pp. 89–104, 1987.

[4] J.-P. Krivine and O. Jehl, "The AUSTRAL system for diagnosis and power restauration: an overview," in *International Conference on Intelligent System Application (ISAP'96)*, (Orlando, USA), Aug. 1996.

[5] S. Bibas, M. O. Cordier, P. Dague, F. Lévy, and L. Rozé, "Scenario generation for telecommunication network supervision," *Workshop on AI in Distributed Information Networks*, Aug. 1995. Montréal, Québec, Canada.

[6] P. Doherty, G. Granlund, K. Kuchcinski, E. Sandewall, K. Nordberg, E. Skarman, and J. Wiklund, "The WITAS unmanned aerial vehicle project," in *Proc. of the 14^{th} ECAI*, (Berlin, Germany), pp. 747–755, Werner Horn, Aug. 2000.

[7] F. Heintz, "Chronicle recognition in the WITAS UAV project – a preliminary report," *Swedish AI Society Workshop (SAIS2001)*, 2001.

[8] J. Gamper and W. Nejdl, "Proposing measurements in dynamic sytems," *Proc. of the 14^{th} IJCAI*, pp. 784–790, Aug. 1995.

[9] M. Dojat, N. Ramaux, and D. Fontaine, "Scenario recognition for temporal reasoning in medical domains.," *Artificial Intelligence in Medecine*, pp. 139–155, 1998.

[10] G. Carrault, M. Cordier, R. Quiniou, M. Garreau, J. Bellanger, and A. Bardou, "A model-based approach for learning to identify cardiac arrhythmias," *Artificial Intelligence in Medecine and Medical Decision Making*, vol. 1620, pp. 165–174, 1999. W. Horn et al. editors.

[11] M. O. Cordier and C. Dousson, "Alarm Driven Monitoring Based on Chronicles," in *Proc. of the 4^{th} Symposium on Fault Detection Supervision and Safety for Technical Processes (SAFEPROCESS)*, (Budapest, Hungary), pp. 286–291, IFAC, A.M. Eldemayer., June 2000.