R. Kervarc
(ONERA)

A. Piel
(CEA LIST, Executable Language Engineering and Optimisation Laboratory R&D Department)

E-mail: romain.kervarc@onera.fr

DOI: 10.12762/2020.AL15-02

A Survey on Chronicles and other Behavior Detection Techniques

Until recently, the processing of a rapidly changing dataflow used to be very costly in terms of computation duration. Thus, the extraction of semantic information, requiring complex correlations of events in a temporal pattern, was not possible in real time. Computer performance has sufficiently improved to now allow such processing to take place, with, obviously, a very broad range of interesting applications. Various underlying formal frameworks exist to perform this kind of analysis, and this paper is aimed at reviewing the various (families of) such formalisms, and at comparing them according to a series of general features. It also provides a detailed example of the kind of analysis that can be performed with these formalisms in aeronautics, where the recent evolution of air traffic management, the potential introduction of unmanned aircraft in general air traffic, and various other recent trends form a general context pointing toward a much wider use of dataflow exchanges.

Introduction

Over the past years, the handling of rapidly changing dataflows at a semantic level has attracted a lot of interest. Indeed, while the semantic processing of a large, fast-pace flow of data used to be too costly in terms of computation, thus obliging to a choice between online syntactic processing and offline semantic processing, computer performance now allows semantic information to be extracted on-the-fly from a large dataflow.

This is, of course, a quite interesting feature in a very broad spectrum of fields, as is evidenced by many recent applications of artificial intelligence. Aeronautics is a particularly interesting field for these dataflow extraction techniques: indeed, while exchanges between pilots and control used to rely mostly on radio, nowadays a large flow of data is exchanged between them. The potential introduction of unmanned aircraft and the recent evolution of air traffic management also point toward a large flow of data exchanges, to which agents of the system have partial access (each having different sensors to track the underlying events), and the introduction of reasoning and semantic processing of these events is a valuable assistance for the pilots and controllers involved.

Traditionally, information extraction from dataflows has been roughly classified into two families of approaches:

- on the one hand, *information flow processing* (IFP): these approaches focus on efficiently handling dataflows by treating incoming information on the fly and providing extracted information in real time;
- on the other hand, *knowledge representation and reasoning* (KRR): these approaches focus on complex reasoning abilities, but perform well mostly on data that changes in low volumes at low frequency.

Stream Reasoning is a multidisciplinary approach to this issue, which encompasses both families and is aimed at combining their respective benefits by enabling complex reasoning about rapidly-changing information flows.

Knowledge-representation and reasoning approaches are based on temporal logic, belief revision, changing vocabularies and evolving ontologies. They find lots of applications within the context of the Semantic Web and allow very complex reasoning tasks. However, there are two drawbacks with regard to these for handling rapidly-changing dataflows. First, the tools involved generally rely on strong combinatorics, and are often not able to scale up to high-frequency dataflows. Second, typical Semantic Web architectures generally crawl and cache information, which is not a robust approach in the case of high-frequency dataflows, since the crawled and cached information would become obsolete too quickly.

Information flow processing is rooted in the so-called Data Stream Management Systems (DSMS), which historically stem from Data Base Management Systems (DBMS). As their name indicates, DBMS are intended to manage databases; i.e., persistent data, where updates are infrequent and where information is extracted through user-made queries. DSMS try to accommodate, within this transient framework, continuously updating data: instead of handling queries that are run just once and of returning a comprehensive answer, DMSM continuously run *standing queries* and return partial answers that are updated on the fly as new data arrives.

Even if they do not seem to have much in common (DBMS handle persistent data by executing ad-hoc queries just once, while DSMS handle transient data by continuously running generic queries), both approaches share a common background and, in particular, process data through transformations based on a relational algebra (e.g., selection, aggregation, joining, etc.). Thus, DSMS can be described, in general, as having very good performance in terms of efficiency, but a rather limited expressivity.

To overcome this limitation, other approaches have been developed in various communities, in which a notion of "distributed system" existed. These approaches have a common characteristic, which is that they consider incoming information in the dataflow not *per se,* but rather as a notification of events occurring in the real world, and are aimed at reconstructing the higher-level behavior of which these events are a trace, mostly through filtering and combinations. In this sense, they are pretty much inspired by the publish-subscribe model that is commonly found in distributed systems: on top of the usual publish-subscribe system, where events are considered separately from the others, they build a more expressive subscription language that allows complex event patterns involving (much) more than one event to be considered. These approaches are referred to by the generic designation of *complex event processing* (CEP). While traditionally classified as part of the IFP family, some CEP techniques have reached reasoning abilities that are comparable to some KRR approaches.

CEP techniques, due to the variety of applications and associated specific needs, exist in a broad variety. This paper is aimed at comparing several CEP frameworks, namely:

- Event Calculus: an approach based on situation calculus, but dealing with local rather than global events;
- ETALIS: an approach based on logical programming and aimed at combining temporal properties with database querying;
- Chronicles: a generic term encompassing various systems based on event signatures;
- Other approaches that are not strictly CEP, but rather based on active databases, DSMS, or KRR.

This paper is organized as follows. First of all, we describe the various important features that can be used to distinguish the various formal frameworks that exist in the CEP community. Then, we discuss the compared merits of the techniques (or technique families) described above, according to these features. We then illustrate the interest of the approach within an aeronautic context, considering an example. Finally, we conclude this survey by providing a brief overview of the various domains where CEP-related techniques have been used with success.

Important features

As stated above, there are many different approaches to stream reasoning and they fulfil the various needs of a broad variety of applications. This section lists the various features that can be used as distinctive criteria for different stream-reasoning approaches.

Language-related features

Any stream-reasoning technique relies upon a formal language that is used to describe the behaviors to be identified. As is frequently the case, expressivity generally results from a trade-off:

- on the one hand, high expressivity is desirable in order to be able to finely describe the behaviors to be detected, and to distinguish behaviors that have very similar traces in terms of observable events;
- on the other hand, higher expressivity inevitably implies a more complex recognition process, and thus less-efficient computation.

In [33] (Section 3.8), Cugola and Margara list operators that are commonly found in the constructs of most CEP frameworks, notably:

- Sequence: two patterns following each other, generally with the sole condition that the first pattern must have been completely recognized before the recognition of the second one starts (some frameworks add that the ending point of the first one must coincide strictly with the starting point of the second one);
- Disjunction: either one of two patterns must be present; when dealing with a rich event (with valued attributes), this can lead to complications because, unless both patterns in the disjunction contain the same event attributes, it may no longer be possible to reason about these attributes;
- Conjunction: it is worth noting that the conjunction cannot be reduced with the two previous operators, since when the elements of conjunction are not elementary events, but rather involve more than one event, conjunction allows intertwined behavior which sequences do not;
- Iteration, which can be parameterized by the number of iterations (which is a parameter relating to the structure of the behavior);
- Negation: this operator is generally tricky when dealing with a flow of information. Indeed, there has to be some form of boundary on the part of the flow, in which a negation must be detected in order to be able to yield effective detections. Otherwise, there will always be a possibility that a later event may trigger the recognition of the negated behavior, and hence invalidate the recognition of its negation. Therefore, many authors prefer to refer to *absence* (implicitly on a bounded support) rather than negation;
- Temporal constraints: while many other temporal logics allow interval properties (e.g., Duration Calculus in [26]) to be expressed, the formalism of Allen's 13 relations [1] is a generally accepted reference in the domain of CEP (see [5, 51, 69]), since it exhaustively considers all possible arrangements between two time intervals;
- Parameter value constraints: the parameters here are parameters of elementary events or of higher-order behaviors.

In addition to the extent to which each of these operators can be expressed, another trait related to the underlying language is the question of whether an open or closed syntax should be used; i.e., whether to allow meaningless formulae or not.

Besides expressivity, an important not-unrelated issue in stream reasoning is how behaviors to be detected can be extracted from experts having an operational knowledge of the behaviors of interest. Within this context, a concise and readable language, where syntactical changes are easily associated with their semantics, is clearly desirable, but again there is a trade-off here, since a very-high readability could lead to an extreme oversimplification of the language, and thus to reduced expressivity.

Recognition-related features

The amount of information contained in recognition is also an important feature, as since recognitions are, in principle, done with a purpose, which implies processing that may require being able to return to the triggering events in the flow. In this sense, recognition information can constitute *evidence* of the recognition, and the nature of this evidence may depend on the application: for example, in some applications, one may be only interested in knowing that the behavior of interest occurred at least once. In other applications, e.g., telecommunication network monitoring [40], a single occurrence of a given behavior is generally not significant, whereas its repetition is. When dealing with security and safety, it is generally necessary to investigate all instances of hazardous behaviors, and not just any. However, another trade-off arises, since the identification of multiple recognitions requires keeping track of all possible recognition starts, and of all intermediate recognitions, thus having an adverse effect with regard to the efficiency of the recognition.

Thus, two main issues arise with regard to recognition: historization and multiplicity.

As explained above, historization is a feature where events are considered as a trace of the behaviors that have been recognized, and recognitions contain the necessary information to return to the events that triggered it.

As to multiplicity, a classification of recognition contexts with respect to this issue has been proposed in [33]:

- The "recent" context: only the most recent occurrence of an event initiating a recognition is kept (each pattern to be recognized is associated with a unique instance of its initiating event at any point during the processing of the flow);
- The "chronicle" context: occurrences are managed in a FIFO way, with the oldest occurrences being used first and discarded as soon as they have been used;
- The "cumulative" context: all event occurrences are stored but, whenever a pattern is recognized, all event occurrences involved are discarded;
- The "continuous" context: all events are stored and can always be used.

Flow-related features

An important feature of any stream-reasoning framework is the way in which it deals with the event flow, and, indeed, which assumptions it makes with regard to it, which determine what kinds of flows it is able to handle. An ideal flow would be a unique flow, totally and strictly ordered: while this kind of flow would probably be handled effectively by most recognition algorithms, it may be possible, depending on the context of the study, to consider other kinds of flows, and notably:

- Distributed flows: the flow may not be centralized, but rather be made of several distributed subflows – in this case, the ordering between events provided by different subflows could be a problem for behavior patterns depending on event order (e.g., sequence), especially if they are not timestamped by a synchronous clocka;
- Partially-ordered flows: events in the flow are not totally ordered, which is a generalization of the previous case and leads to the same problems;
- Non-strictly ordered flows: events may arise simultaneously; this is a case that may have unexpected side-effects in situations where event order is important for the behavior;
- Delayed flow: events may arise late, or their occurrence date may be corrected *a posteriori* (which happens, e.g., in cases where events may be revised following the failure of a transaction) – in general, the date on which the event occurs differs from the date on which it is entered into the reasoning system, which requires specific mechanisms to handle events properly.

In addition, another important aspect of the flow is the time model that it uses. Time is generally linear, but can be discrete, either with a fixed pace or with variable granularity, or continuous. The time model may also not exist: this is the case in most DSMS, where there is no time model and events are considered only with an order – but this has, of course, a negative impact on the expressivity of the language.

Uncertainty-related features

In real cases, lots of uncertainties appear naturally:

- With regard to event dates (or order);
- With regard to event attributes;
- With regard to the events themselves (whether they really occurred or not);
- With regard to behavior parameters;
- With regard to behavior structures.

Depending on the uncertainties considered, specific mechanisms have to be considered, either in the description language or in the recognition algorithm itself.

Self-reference features

Recognitions can be self-referent and create events at various points of the recognition process. The most frequent occurrence of this feature is when the recognition of a behavior triggers a new event that is added to the flow. This yields many issues, since such systems are intrusive: in particular, the flow depends on the monitored activities. In extreme cases, recognitions may also have an effect on the monitored activities, where new activities to be recognized are dynamically added into the flow, leading to a retroaction loop that is difficult to manage, both theoretically and practically.

Event Calculus

Event Calculus (EC) is a formal framework allowing events and actions to be represented and reasoned upon in the form of an executable logical program. It is aimed at determining time-evolved values for logical propositions (the so-called *fluents*).

EC was introduced by Kowalski and Sergot in [50]. Its name is derived from Situation Calculus; the difference between these two frameworks being that it deals with local, rather than global, events:

the purpose of this change is to avoid the frame problem for the sake of efficiency. EC claims to provide a formal analysis of the concepts involved; i.e., events and actions. It can be expressed using Horn clauses, to which, consistently with the logical programming foundation of the approach, a notion of negation by failure is added (thus introducing a closed-world assumption).

The founding principles of EC are the following:

- Events can be processed in any order, not necessarily in relationship with their order of occurrence, since the past and future are considered symmetrically;
- Events can be concurrent and are not necessarily punctual (an elementary event can have a duration);
- Updates are possible, but only if they are additive: they can add information but never remove information;
- The dates of events are not particularly relevant, whereas their relative order is.

Many EC dialects exist, some of which allow the handling of delayed actions or continuous state changes, such as, e.g. [58, 57], an interval-based work built with reaction rules; these approaches are catalogued in [54].

An interesting modular approach can be found in the EC dialect developed by Artikis et al.: this dialect allows a low-level event to be composed into high-level complex behaviors, using the predicates found in Table 1 to express temporal constraints, with an underlying linear time model. These predicates are defined relying upon axioms, some of which may be independent from the application domain. The formalism is quite expressive, and allows constraints, whether temporal or not, to be expressed and contains a form of absence, in the form of a situation where a given behavior must not occur within a certain time interval [9]. High-level behaviors can be defined using punctual events (through predicate `happensAt`) or fluents, initially using predicates such as `initiatedAt`, `holdsFor`, etc.

Predicate	Intuitive meaning
`happensAt`(E, T)	Event E occurs at time T.
`initially`$(F = V)$	Fluent F has value V at time 0.
`holdsAt`$(F = V, T)$	Fluent F has value V at time T .
`holdsFor`$(F = V, I)$	I is the list of all maximum time intervals over which F has value V.
`initiatedAt`$(F = V, T)$	A time interval where F has value V starts at time T.
`terminatedAt`$(F = V, T)$	A time interval where F has value V ends at time T.
`union_all`(L, I)	I is the list of all maximum time intervals resulting from the union of all intervals in List L.
`intersect_all`(L, I)	I is the list of all maximum time intervals resulting from the intersection of all intervals in List L.
`relative_complement_all`(I', L, I)	I is list It of maximum time intervals minus each set of intervals in List L.

Table 1 - Main predicates of Event Calculus

The issue of behavior extraction and writing is studied in [13, 8]. Indeed, writing activities in the EC framework is tedious and error-prone, hence the idea of developing an automated process to generate definitions from temporal data. Thus, the authors use a learning method based on abductions and inductions to infer the behaviors to be recognized.

As explained in the introduction to this article, a major issue for stream reasoning is whether behaviors can be recognized or not in real time. The algorithm in this EC dialect uses a system query method: the reasoning is not performed gradually, but rather on demand, whenever a high-level activity is queried [10]. Thus, in order to perform an online analysis, it is necessary to constantly make queries: without a cache, this implies starting computations over again each time. Moreover, one of the principles of AC is that the order in which events occur further increases the complexity of the computation. In [28], Chittaro et al. introduce a version of EC called Cached Event Calculus (CEC), an implementation managing a cache memory to reduce the complexity of the process. However, CEC has no pre-emption mechanism and it accepts the processing of events with an earlier date than already processed events.

Therefore, recognition times increase gradually as low-level events occur, and after the computation becomes too time-expensive to keep up in real time. Artikis et al. attempted to address this issue in [12], where they introduce RTEC (Run-Time reasoning Event Calculus), an efficient YAProlog[1] implementation of their EC. Their program is also based on successive queries, with a cache memory preserving maximum intervals computed for the HoldsFor predicates of each fluent.

In addition, in order to address uncertainty, several stochastic approaches of EC have been developed. Artikis et al. extended the formalism of EC in [65] by means of Markov logical networks (MLN) [38], combining first-order logic with the probabilistic semantics of Markov networks. In [64] they also provided another extension, this time to probabilistic logical programming, using Prob-Log [49]. This way, they addressed the issue of incorrect low-level event detections by adding confidence indices to the events in the flow. The uncertainty here is limited to event uncertainty: in particular, it is not possible to handle incompletely specified behaviors. Moreover, the authors admit that online recognition is not possible in this formalism, which is corroborated in a recent work by Rincé et al. [63], who showed that, for a whole class of problems, the local search algorithms necessary in MLN and ProbLog perform poorly due to the structural characteristics of the problem.

Another approach for uncertainty handling is introduced in [14]. It is orthogonal to that in [64] in the sense that they may be combined. It relies upon the use of various event sources to determine their likelihood, with an auto-adaptation system based on the behavior recognition process itself: complex behavior definitions are written to identify the uncertainty domains and react accordingly: when the uncertainty becomes significant, the system may ignore events over a certain time interval, or even momentarily discard an event source. In [15], the authors add crowdsourcing to this framework, in order to make decisions when discrepancies between sources become significant.

ETALIS

Event-driven Transaction Logic Inference System (ETALIS)[2] [5, 2] is a CEP language, the syntax and semantics of which allow reasoning simultaneously on temporal assertions and on stable or evolving knowledge (rules, facts, ontologies, encyclopedic data, etc.). Its processing engine allows behaviors to be analyzed online.

[1] http://www.dcc.fc.up.pt/~vsc/Yap/

[2] available in open-source at http://code.google.com/p/etalis/

ETALIS is a logical programming language, and its syntax is defined by rules, the main constructs of which are shown in Table 2. The underlying time model is linear, dense, but countable (i.e., $\mathbb{Q}$), and low-level events may be instantaneous events, as well as events with a duration: events are dated by time intervals $[T_1, T_2]$ (with $T_1 = T_2$ in the case of instantaneous events). The language has a high expressiveness and contains:

- all of Allen's 13 interval relations;
- constraints on event properties;
- a rather limited notion of absence within the framework of the sequence;
- two precisely distinguished kinds of conjunction (in series and in parallel);
- recursive behavior definitions, allowing, for example, the definition of a function accumulating a value over a sequence of events.

A first formal declarative semantic approach is provided in [3, 5], where event patterns (i.e., behaviors) are defined by induction in the manner of model theory. A recognition is a couple (q_1, q_2), with q_1, $q_1 \in \mathrm{Q}$ delimiting the necessary and sufficient time interval for the recognition (its support). Other than this support, in which they must all have been encompassed, information pertaining to the events triggering the recognition is not kept: there is no possibility of historization, and multiplicity is limited to the cases in which the supports of the multiple recognitions are distinct.

Constructs	Intuitive meaning
p WHERE t	Behavior p has been recognized and the term t is valued to true.
q	This corresponds to the absolute instant q (for any $q \in \mathbb{Q}$).
$(p).q$	Behavior p has been recognized and lasts exactly q, with $q \in \mathbb{Q}$.
p_1 SEQ p_2	Behavior p_1 is strictly followed (in time) by behavior p_2.
p_1 AND p_2	Behaviors p_1 and p_2 have been recognized, without any temporal constraint.
p_1 PAR p_2	Behaviors p_1 and p_2 have been recognized in parallel; i.e., they overlap in time.
p_1 OR p_2	Either one of both behaviors has been recognized.
p_1 EQUALS p_2	Both behaviors have been recognized over the exact same time interval.
p_1 MEETS p_2	Both p_1 and p_2 have been recognized, and the last recognition instant for p_1 exactly matches the first recognition instant of p_2.
p_1 DURING p_2	Behavior p_1 has been recognized within the recognition of p_2.
p_1 STARTS p_2	The recognition interval of p_1 is an initial segment of the recognition interval of p_2.
p_1 FINISHES p_2	The recognition interval of p_1 is a final segment of the recognition interval of p_2.
NOT $(p_1).[p_2, p_3]$	Behaviors p_2 and p_3 have been recognized in this order, without any occurrence of p_1 strictly contained between both in time.

Table 2 - Main constructs of ETALIS [3]

The ETALIS recognition system is implemented in Prolog. This implementation relies on an operational semantics defined using logic programming rules. The complex behaviors to be recognized are are broken up into intermediate events called *goals*. ETALIS compiles complex behaviors into a set of rules allowing Event-Driven Backward Chaining, which allows an online recognition process. Two types of rules result from the compilation:

- rules creating the goals to be recognized, in order to progress in the recognition of a complex behavior, in the form of an event and the expectation of another event: `goal`$\left(b^{[-,-]}, a^{[T_1,T_2]}, ie_1^{[-,-]}\right)$ means that when a (potentially complex) behavior a has been recognized over interval $[T_1, T_2]$, the system expects an event b to recognize behavior ie_1;
- rules creating intermediate events or event patterns: these check the database to determine whether a certain goal already exists, and, if this is the case, trigger the event that has been recognized by the goal: if `goal`$\left(b^{[T_3,T_4]}, a^{[T_1,T_2]}, ie_1^{[-,-]}\right)$ is in the database, then event $ie_1^{[T_1,T_4]}$ is triggered and propagated if it is an intermediate event, or is used to trigger an action if it is one of the complex behaviors sought.

Rules of the latter type also allow goals that are obsolete and not needed anymore to be suppressed from the database.

In other terms, the underlying recognition structure is a binary tree. However, the equivalence of both these semantics has not been proven.

As to recognition multiplicity, ETALIS allows the following event consumption policies: recent, chronicle, and "free" (i.e., without any restriction). However, the declarative aspect is lost with any policy other than free, which means that the rule-evaluation order ceases to be neutral.

The performance of ETALIS is also assessed on a so-called Fast Flower Delivery use case [43].

ETALIS also handles delayed events [44] through two additional rule types:

- `goal_out`$\left(a^{[-,-]}, b^{[T_3,T_4]}, ie_1^{[-,-]}\right)$, expressing that Event b has been received and that an Event a having occurred *before* b is expected to finalize the recognition of ie_1.
- `if goal_out(...)` and $T_2 < T_3$, expressing that if an event a indeed occurs at $T_2 < T_3$, then event $ie_1^{[T_1,T_4]}$ is triggered.

This algorithm does not have adverse effects on the efficiency of the recognition for events occurring on time. However, it requires a specific procedure to free memory by suppressing `goal_out` rules after a while. Reference [44] explains that, due to practical reasons (probably a matter of recognition efficiency by preventing rule overload) this functionality has not been implemented: therefore, multiplicity is lost. To handle delayed events in the case of an absence, ETALIS also allows the handling of *revised* events [4]: new rev goals are introduced to suppress revised goals.

Chronicles

Chronicles are a family of formal languages developed to formally describe an event signature and, as such, provide a framework for CEP.

Dousson's chronicles

A chronicle language was introduced in [45], and developed mainly by Dousson *et al.* [39, 40, 41]. Within this framework, a chronicle is somehow a partial order of observable events in a certain context. Together with the language comes an efficient online recognition process enabling the analysis of a flow of timestamped events that do not necessarily arrive in their order of occurrence, with the possibility of triggering actions or producing events at a date defined in relation to the dates of the events having caused the recognition.

In [39]: a *chronicle model* is presented as a set of formulae or temporal schemas defining how the association of several *observable* events can lead to a new *deduced* event, and a set of constraints is given. A chronicle is thus a set of events together with contextual and temporal constraints. In this approach, the time model is discrete, totally ordered, and precise enough to take into account the observed events. In more recent works, Dousson *et al.* [41] associate attributes with events that can change their values: chronicles are represented by constraint graphs, with events as nodes, and the edges are labelled with integer intervals that represent time constraints.

This framework has also been used and adapted by Subias and Boufaied [18, 46] to various contexts, but always with discrete time. In his PhD work [42], Vu Dũ'o'ng applies Dousson's work to telecommunication network diagnosis through alarm correlation.

Here is a broad idea of how chronicles are expressed in Dousson's formalism. They are multi-sets of events with additional constraints expressed as time intervals (which may contain negative values, meaning that the events occur in reverse order than that specified) that must be fulfilled by pairs of events. For instance (see Figure 1), a chronicle may be $ABCD$ where the interval between A and C must be within $[-3, 2]$, the interval between A and D must be within $[4, 6]$, and the interval between D and B must be within $[-1, 4]$. Each event in the multi-set has to be mapped exactly once to an event of the flow, and the mapping must be consistent with the constraints.

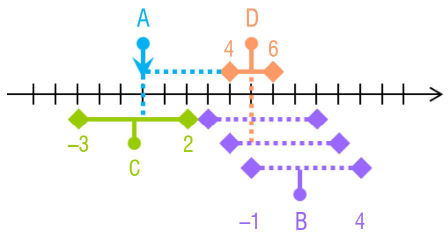

Figure 1 - An example of Dousson's Chronicle: $C \xleftarrow{[-3,2]} A \xrightarrow{[4,6]} D \xrightarrow{[-1,4]} B$

The expressivity of Dousson's formalism is rather low. In particular, the fact that chronicles can combine various intermediate patterns that may or may not share elementary events cannot be expressed. For instance, it is not possible to write a chronicle of the form $(ABC)\&(DBE)$, that would be recognized if an event B occurs between A and C, and an event B (possibly the same, but not necessarily) occurs between D and E. This cannot be expressed in Dousson's formalism, where it is necessary to specify when designing the chronicle whether there is a single B or two distinct B. In addition, absence is difficult to account for in Dousson's work. Moreover, the issue above regarding shared events between subchronicles also applies for absence.

ONERA chronicles

In the formalism of ONERA chronicles, events are represented as ordered pairs comprising an event name and a real number (its occurrence *date*). The underlying time model is linear and continuous. These events can be endowed with information, called *attributes*, which are ordered pairs of an *attribute name* and a *value*.

Attributes are a very expressive feature of the language: an event can have any number of attributes and, given that recognitions are built upon events, new attributes can be computed, named and associated with recognitions, so as to be used at a higher level.

The chronicle language is built by induction, using, among others, four constructs expressing the sequence, the conjunction, and the disjunction of two behaviors, as well as the absence of a given behavior during another behavior. These constructs have been presented in [20, 21].

In addition to this, ONERA chronicles express all of Allen's 13 relations, as well as constraints on the durations of behaviors and a few additional constructs, such as a change of state and a derived event associated with the instant of recognition completion. Moreover, the chronicle language allows reasoning on *event attributes*: a predicate can express desired constraints on manipulated attributes. Constraints on attributes and attribute creations can be added at each level, and a notion of the evaluation context allows attributes to be handled properly in constructs where not all subchronicles are present in the recognition of a chronicle (typically, absence and disjunction).

The notion of chronicle recognition, originally [20, 21] based on a notion of a set of events leading to it, has therefore been replaced by a tree-based notion: indeed, an event model set does not retain the information specifying which event led to the recognition of which sub-chronicle, which becomes an issue when properties are expressed over event attributes. Consider, for example, Chronicle $C = (A\ B)\ \&\ A$. Some recognitions of C may be due to two distinct events a (a denotes an instance of A). In a set formalism, these two events are undistinguishable, so it is impossible to determine which *a* led to the recognition of sub-chronicle $A\ B$, and which led to the recognition of the single A. Not only is this information lost, but this also affects the combinatorics, since the number of recognitions depends on this information: if this information is kept, two events a lead to two different recognitions of C depending on the distribution[3].

Continuous time is managed through the use of a look-ahead function $T_C(\varphi, d)$ providing a future date until which the system does not need to be re-examined, since the recognition set would not have changed until then. Indeed, the systems considered here are asynchronous, and this function provides the next time when it will be necessary to check for the completion of a given chronicle with delays.

More details on the theoretical framework of chronicle recognition are presented in [60], but their general form is that of a triplet (C, P, f), where:

- $C \in \mathfrak{X}$ is a *chronicle formula* (see below);
- $P \in \mathfrak{S}$ is a predicate symbol;
- $f \in \mathfrak{T}(\mathfrak{P}, \mathcal{V})$ is an attribute transformation.

[3] Note that there is also one additional recognition for each event a leading to the recognitions of both sub-chronicles.

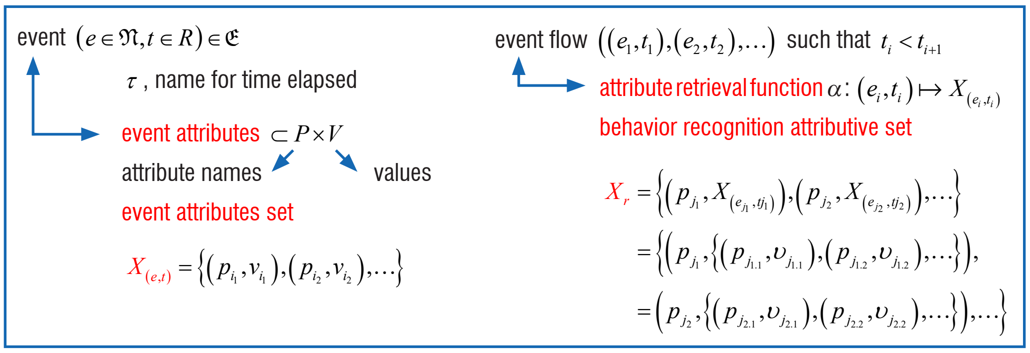
event $(e \in \mathfrak{N}, t \in R) \in \mathfrak{E}$
τ , name for time elapsed
event attributes $\subset P \times V$
attribute names
values
event attributes set
$X_{(e,t)} = \{(p_{i_1}, v_{i_1}), (p_{i_2}, v_{i_2}), \ldots\}$
event flow $((e_1,t_1),(e_2,t_2),\ldots)$ such that $t_i < t_{i+1}$
attribute retrieval function $\alpha : (e_i, t_i) \mapsto X_{(e_i,t_i)}$
behavior recognition attributive set
$X_r = \{(p_{j_1}, X_{(e_{j_1}, tj_1)}), (p_{j_2}, X_{(e_{j_2}, tj_2)}), \ldots\}$
$= \{(p_{j_1}, \{(p_{j_{1.1}}, \upsilon_{j_{1.1}}), (p_{j_{1.2}}, \upsilon_{j_{1.2}}), \ldots\}),$
$= (p_{j_2}, \{(p_{j_{2.1}}, \upsilon_{j_{2.1}}), (p_{j_{2.2}}, \upsilon_{j_{2.2}}), \ldots\}), \ldots\}$

Figure 2 - Events and attributes for chronicles

$\mathfrak{X}$ is inductively defined together with two notions of *contexts*, which are functions from $\mathfrak{X}$ to $\mathfrak{P}$ (an evaluation context $\mathcal{C}_e$ and a resulting context $\mathcal{C}_r$):

simple event: If $A \in \mathfrak{N}$, then $(\mathbf{A}, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(A,P,f) = \{\Diamond\}$, et $\mathcal{C}_r(A,P,f) = \mathcal{C}_e(A,P,f)$;

sequence: If $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, then $(\mathbf{C_1}\ \mathbf{C_2}, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1\ C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, et $\mathcal{C}_r(C_1\ C_2, P, f) = \mathcal{C}_e(C_1\ C_2, P, f)$;

conjunction: If $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, then $(\mathbf{C_1} \,\&\, \mathbf{C_2}, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1 \,\&\, C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, $\mathcal{C}_r(C_1 \,\&\, C_2, P, f) = \mathcal{C}_e(C_1 \,\&\, C_2, P, f)$;

disjunction: $(\mathbf{C_1} \| \mathbf{C_2}, P, f) \in \mathfrak{X}$,$\mathcal{C}_e(C_1 \| C_2, P, f) = \mathcal{C}_r(C_1) \cap \mathcal{C}_r(C_2)$, and $\mathcal{C}_r(C_1 \| C_2, P, f) = \mathcal{C}_e(C_1 \| C_2, P, f)$;

absence: If $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, then $((\mathrm{C}_1) - [\mathrm{C}_2], P, f) \in \mathfrak{X}$, $\mathcal{C}_e((C_1) - [C_2], P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, and $\mathcal{C}_r((C_1) - [C_2], P, f) = \mathcal{C}_r(C_1)$;

meets: If $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, then $(\mathrm{C}_1\, \text{meets}\, \mathrm{C}_2, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1\, \text{meets}\, C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, and $\mathcal{C}_r(C_1\, \text{meets}\, C_2, P, f) = \mathcal{C}_e(C_1\, \text{meets}\, C_2, P, f)$;

overlaps: If $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, then $(\mathrm{C}_1\, \text{overlaps}\, \mathrm{C}_2, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1\, \text{overlaps}\, C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, and $\mathcal{C}_r(C_1\, \text{overlaps}\, C_2, P, f) = \mathcal{C}_e(C_1\, \text{overlaps}\, C_2, P, f)$;

starts: If $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, then $(\mathrm{C}_1\, \text{starts}\, \mathrm{C}_2, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1\, \text{starts}\, C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, and $\mathcal{C}_r(C_1\, \text{starts}\, C_2, P, f) = \mathcal{C}_e(C_1\, \text{starts}\, C_2, P, f)$;

during: If $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, then $(\mathrm{C}_1\, \text{during}\, \mathrm{C}_2, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1\, \text{during}\, C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, and $\mathcal{C}_r(C_1\, \text{during}\, C_2, P, f) = \mathcal{C}_e(C_1\, \text{during}\, C_2, P, f)$;

finishes: If $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, then $(\mathrm{C}_1\, \text{finishes}\, \mathrm{C}_2, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1\, \text{finishes}\, C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, and $\mathcal{C}_r(C_1\, \text{finishes}\, C_2, P, f) = \mathcal{C}_e(C_1\, \text{finishes}\, C_2, P, f)$;

equals: If $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, then $(\mathrm{C}_1\, \text{equals}\, \mathrm{C}_2, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1\, \text{equals}\, C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, and $\mathcal{C}_r(C_1\, \text{equals}\, C_2, P, f) = \mathcal{C}_e(C_1\, \text{equals}\, C_2, P, f)$;

lasts δ: If $\delta \in \mathbb{R}_*^+$, then $(\mathrm{C}_1\, \text{lasts}\, \delta, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1\, \text{lasts}\, \delta, P, f) = \mathcal{C}_r(C_1)$, and $\mathcal{C}_r(C_1\, \text{lasts}\, \delta, P, f) = \mathcal{C}_e(C_1\, \text{lasts}\, \delta, P, f)$;

at least δ: If $\delta \in \mathbb{R}_*^+$, then $(\mathrm{C}_1\, \text{atleast}\, \delta, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1\, \text{atleast}\, \delta, P, f) = \mathcal{C}_r(C_1)$, and $\mathcal{C}_r(C_1\, \text{atleast}\, \delta, P, f) = \mathcal{C}_e(C_1\, \text{atleast}\, \delta, P, f)$;

at most δ: If $\delta \in \mathbb{R}_*^+$, then $(\mathrm{C}_1\, \text{atmost}\, \delta, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1\, \text{atmost}\, \delta, P, f) = \mathcal{C}_r(C_1)$, and $\mathcal{C}_r(C_1\, \text{atmost}\, \delta, P, f) = \mathcal{C}_e(C_1\, \text{atmost}\, \delta, P, f)$;

then δ: If $\delta \in \mathbb{R}_*^+$, then $(\mathrm{C}_1\, \text{then}\, \delta, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1\, \text{then}\, \delta, P, f) = \mathcal{C}_r(C_1)$, and $\mathcal{C}_r(C_1\, \text{then}\, \delta, P, f) = \mathcal{C}_e(C_1\, \text{then}\, \delta, P, f)$;

naming: If $x \in \mathfrak{P} \setminus \{\Diamond\}$, then $(\mathrm{C}_1 \to \mathrm{x}, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1 \to x, P, f) = \mathcal{C}_r(C_1)$, $\mathcal{C}_r(C_1 \to x, P, f) = \{x, \Diamond\}$;

cut: If $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, then $(\mathrm{C}_1 ! \mathrm{C}_2, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1 ! C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, and $\mathcal{C}_r(C_1 ! C_2, P, f) = \mathcal{C}_e(C_1 ! C_2, P, f)$;

change of state: If $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, then $(\mathrm{C}_1 !! \mathrm{C}_2, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(C_1 !! C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, and $\mathcal{C}_r(C_1 !! C_2, P, f) = \mathcal{C}_e(C_1 !! C_2, P, f)$;

recognition event: $(@\mathrm{C}_1, P, f) \in \mathfrak{X}$, $\mathcal{C}_e(@C_1, P, f) = \mathcal{C}_r(C_1)$, and $\mathcal{C}_r(@C_1, P, f) = \mathcal{C}_r(C_1)$.

Chronicles allow for multiplicity and historization of recognitions. They also allow the gradual recognition of behaviors in real time, as events flow. The need for multiplicity and historization prevents the use of simple finite-state automata (see [17]), but a first recognition tool called Chronicle Recognition System (CRS/ONERA) was developed at ONERA in the late 1990s in [19], based on duplicating automata so as to comply with performance and inter- operability requirements. A colored Petri net model was also developed, implementing chronicle recognition for the initial operators – sequence, conjunction, disjunction, and absence – and its adequacy has been proven (see [23, 21]). A new recognition tool has also been developed in the form of a C++ library called Chronicle Recognition Library (CRL)[4], which can be easily used for real-world critical applications. Its algorithms are directly based on the formal semantics of the chronicle language, therefore the recognitions produced by CRL are considered t be adequate by construction. The efficiency of CRL is also ensured by a validity window mechanism that eliminates obsolete initiated recognitions after a time specified by the user. Applications of CRL are presented in [24, 22].

Other approaches

Other families of approaches to stream reasoning are at the two far ends of the spectrum of SR techniques: DSMS and KRR. We illustrate them in this section through two representative examples: CQL (a DSMS framework) and LARS (based on KRR and answer set programming).

Continuous Query Language

Continuous Query Language (CQL) [6, 7] is a language based on the database query language SQL, extended with streams as additional data sources. In CQL, a *stream* is viewed as a bag of elements in the form $\langle c,t \rangle$, where c is a tuple and t is a timestamp; a *relation* maps timestamps to bags of tuples. To make these concepts compatible, the operational semantics of CQL relies on three kinds of operators:

- Relation-to-Relation operators contain usual SQL operators to manipulate relations;
- Stream-to-Relation operators apply window functions to the input stream to create a relation for recent tuples;
- Relation-to-Stream operators translate back a relation into a stream for the output of continuous queries.

There are interesting parallels between CQL and our approach; in particular, the fact that CQL has operational semantics where evaluation is performed stepwise as a query is evaluated. However, being based on SQL-like queries, CQL handles the stream by filtering, joining and aggregating data in a deterministic way, and does not allow for abstractions, constraints, complex negation, and non-determinism.

Compared to CQL, our approach allows additional abstraction and reasoning features, including the absence (which is a form of complex negation) and temporal modalities. Both features are particularly important in our approach, as evidenced by the various levels of reasoning in aerospace case studies that we have treated (cf. Table 5.9 in [60]): indeed, intermediate (i.e., Level-2) and interest (i.e., Level-3) chronicles contain temporal modalities (e.g., at least, @, and !! in Level-2 chronicles, as well as in Level-3 chronicle NoClearanceToTakeOff(*ID*), as well as absences with additional correlations to event attributes; e.g., Level-3 chronicle NoFrequencyToTakeOff(*ID*).

[4] CRL has been deposited at the French Agency for Program Protection and is available under the GNU LGPL license.

LARS

LARS [16] consists of two languages: *LARS formulae* extend propositional logic with generic window operators and additional controls to handle temporal information, and, on top of this, *LARS programs* extend Answer Set Programming (ASP) with rich stream-reasoning capabilities. It is aimed at targeting AI applications in a streaming context, such as diagnosis, configuration, or planning.

Fragments of LARS have been implemented in several experimental prototypes [16], based on different realization principles, but they either lack efficiency or are restricted to specific LARS programs, in particular with restrictions on the use of negations.

In contrast to chronicles, LARS semantics is based on time points. Nevertheless, as stated in [16], when comparing LARS and ETALIS, it is possible to represent intervals in LARS and thus partially capture the notion. However, this representation is unable to take into account overlapping intervals (for a same formula): indeed, LARS assigns atoms to a single timeline by an evaluation function, so it can encode intervals only by assigning atoms to consecutive time points. Adjacent or overlapping intervals for the same atom cannot be distinguished and, worse still, merge into a single larger interval, which is incompatible with our objective of multiple recognitions.

An application to aeronautics

To illustrate the interest of these techniques in the aeronautic field, we present an example of a hypothetical unmanned aircraft inserted into general air traffic, as described in Figure 3.

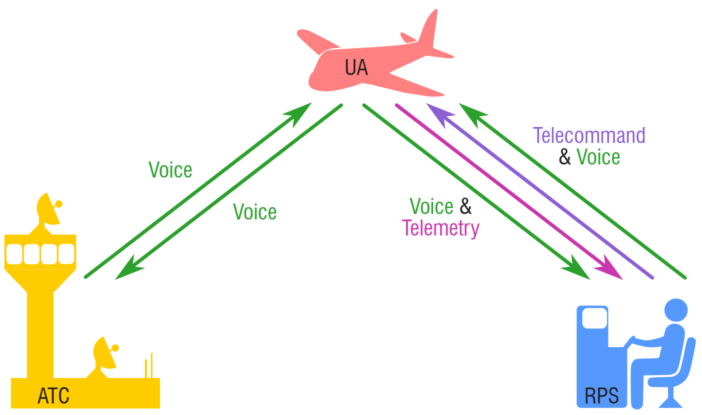

Figure 3 - Schematic representation of the three-agent system

This is a global problem that raises many interesting issues, and implies interactions between the aircraft, its pilot (on the ground) and Air Traffic Control (ATC). We focus here on a potential hazard, which is the loss of the telecommand (TC) link, meaning that the pilot is unable to transmit orders to the unmanned aircraft. In such a case, there has to be a predetermined course of action (e.g., return to base, pursue current route, land at the nearest airport, etc., – which one exactly is not relevant here). If such a loss occurs, it is obviously important that all three agents (ATC, pilot, and aircraft) share the same understanding of the situation, so that, in particular, both of the human actors act consistently.

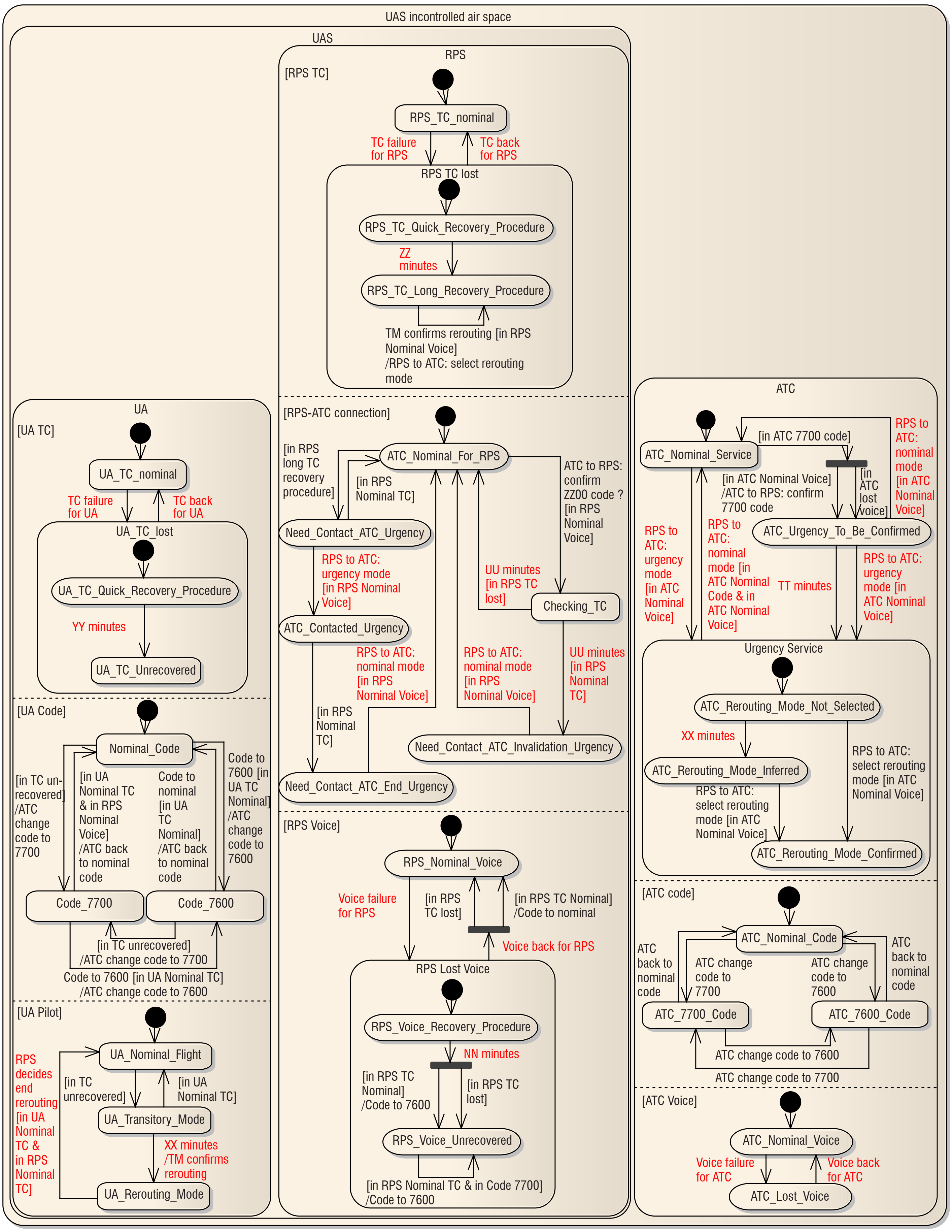
UAS incontrolled air space
UAS
RPS
[RPS TC]
RPS_TC_nominal
TC failure for RPS
TC back for RPS
RPS TC lost
RPS_TC_Quick_Recovery_Procedure
ZZ minutes
RPS_TC_Long_Recovery_Procedure
TM confirms rerouting [in RPS Nominal Voice] /RPS to ATC: select rerouting mode
[RPS-ATC connection]
ATC_Nominal_For_RPS
[in RPS long TC recovery procedure]
[in RPS Nominal TC]
ATC to RPS: confirm ZZ00 code ? [in RPS Nominal Voice]
Need_Contact_ATC_Urgency
RPS to ATC: urgency mode [in RPS Nominal Voice]
UU minutes [in RPS TC lost]
Checking_TC
ATC_Contacted_Urgency
RPS to ATC: nominal mode [in RPS Nominal Voice]
RPS to ATC: nominal mode [in RPS Nominal Voice]
UU minutes [in RPS Nominal TC]
[in RPS Nominal TC]
Need_Contact_ATC_Invalidation_Urgency
Need_Contact_ATC_End_Urgency
[RPS Voice]
RPS_Nominal_Voice
Voice failure for RPS
[in RPS TC lost]
[in RPS TC Nominal] /Code to nominal
Voice back for RPS
RPS Lost Voice
RPS_Voice_Recovery_Procedure
NN minutes
[in RPS TC Nominal] /Code to 7600
[in RPS TC lost]
RPS_Voice_Unrecovered
[in RPS Nominal TC & in Code 7700] /Code to 7600
UA
[UA TC]
UA_TC_nominal
TC failure for UA
TC back for UA
UA_TC_lost
UA_TC_Quick_Recovery_Procedure
YY minutes
UA_TC_Unrecovered
[UA Code]
Nominal_Code
[in TC un-recovered] /ATC change code to 7700
[in UA Nominal TC & in RPS Nominal Voice] /ATC back to nominal code
Code to nominal [in UA TC Nominal] /ATC back to nominal code
Code to 7600 [in UA TC Nominal] /ATC change code to 7600
Code_7700
Code_7600
[in TC unrecovered] /ATC change code to 7700
Code to 7600 [in UA Nominal TC] /ATC change code to 7600
[UA Pilot]
UA_Nominal_Flight
RPS decides end rerouting [in UA Nominal TC & in RPS Nominal TC]
[in TC unrecovered]
[in UA Nominal TC]
UA_Transitory_Mode
XX minutes /TM confirms rerouting
UA_Rerouting_Mode
ATC
ATC_Nominal_Service
[in ATC 7700 code]
RPS to ATC: nominal mode [in ATC Nominal Voice]
[in ATC Nominal Voice] /ATC to RPS: confirm 7700 code
[in ATC lost voice]
RPS to ATC: urgency mode [in ATC Nominal Voice]
RPS to ATC: nominal mode [in ATC Nominal Code & in ATC Nominal Voice]
ATC_Urgency_To_Be_Confirmed
TT minutes
RPS to ATC: urgency mode [in ATC Nominal Voice]
Urgency Service
ATC_Rerouting_Mode_Not_Selected
XX minutes
RPS to ATC: select rerouting mode [in ATC Nominal Voice]
ATC_Rerouting_Mode_Inferred
RPS to ATC: select rerouting mode [in ATC Nominal Voice]
ATC_Rerouting_Mode_Confirmed
[ATC code]
ATC_Nominal_Code
ATC back to nominal code
ATC change code to 7700
ATC change code to 7600
ATC back to nominal code
ATC_7700_Code
ATC_7600_Code
ATC change code to 7600
ATC change code to 7700
[ATC Voice]
ATC_Nominal_Voice
Voice failure for ATC
Voice back for ATC
ATC_Lost_Voice

Figure 4 - State diagram of telecommand loss

However, each actor has access only to a partial subset of information, from which they deduce the status of the TC link. Hence, their reasoning can be modelled through chronicles and this can be used to detect inconsistencies.

Here, low-level observable events will schematically be the actions performed by each agent on the system or their changes of state, which are modelled according the state diagram in Figure 4 representing the protocol followed by each agent. The diagram is distributed between the three agents: UA (Unmanned Aircraft), RPS (Remote Pilot Station), ATC. Each agent is broken down into sub-systems. A sub-system represents a functionality controlled by the agent, or a specific knowledge that it may possess about the overall system situation; e.g., the RPS Voice sub-system describes whether the RPS is aware of a potential radio communication loss. Each sub-system is a set of states that have to be followed in a specific order regarding the system evolution. Arrows between states describe this order and the necessary conditions to trigger a state change, and may possibly be associated with a specific action to be performed by the agent. This information is labelled on the arrows in three different parts: *event*, *condition* and *action,* and written `event [condition]/action`.

The formalism can then be used to detect undesired behaviors, such as:

- *Incoherent ATC Voice*: the transponder code emitted by the UA starts indicating code 7600 to Air Traffic Control, which means that there is a voice failure, but the controller has not realized this, and this is expressed by the fact that the diagram does not switch to `ATC Lost Voice`.
- *Incoherent flight mode UA/ATC*: after a fault that has been solved, the UA has switched back to a nominal flight but ATC remains in an urgency service.

Once one of these behaviors is detected, its origins have to be determined. If the cause is due to faulty behavioral guidelines, then the model has to be corrected, and, otherwise, if the source is human, it should be planned to trigger alarms warning the pilot and/or the air traffic controller of the situation.

These behaviors are represented by the following chronicles:

- *Incoherent ATC Voice*

(`to_ATC_Nominal_Code to_ATC_7600_Code` then 5) – [`to_ATC_Lost_Voice`]

- *Incoherent flight mode UA/ATC*

(`from_UA_Nominal_Flight`
((`to_UA_Nominal_Flight` then 10) – [`from_UA_Nominal_Flight`]))
–[`to_ATC_Nominal_Service`]

The possibility, once a hazardous behavior has been recognized, of determining its origin is provided by the properties of the chronicle framework.

Conclusion

While no single Stream-Reasoning approach can claim to be able to tackle all possible uses of Stream Reasoning, the overview of methods presented in this paper shows that handling a rapidly changing dynamic dataflow with elaborate reasoning is now feasible. These methods find applications in a very broad spectrum, which includes:

- the detection of inconsistencies between pilot and air traffic control in a scenario where an unmanned aircraft may lose its telecommand [24] (see above), with chronicles;
- the supervision and analysis of hazardous situations using an unmanned aircraft to assist police services [47, 35, 36, 34], with chronicles;
- various medical applications, including heart monitoring [30, 25, 62, 37, 61, 27], using chronicles together with learning techniques;
- management of alarms for the detection of cyber-intrusions [56], with Dousson's chronicles;
- Web-service diagnostics [59, 31, 52], with chronicles;
- public transportation quality assessment [48, 68], with EC (project PRONTO);
- video-surveillance [66, 13, 11, 9], with EC (project CAVIAR);
- social media analysis [66];
- assistance in decision-making during air combat [29], with chronicles;
- network supervision and monitoring management[67], with chronicles;
- supervision of a gas turbine in a petrochemical plant [55] and supervision of a milk factory [53], with chronicles;
- characterization of human activities [32], with chronicles ■

References

[1] J. F. ALLEN - *Maintaining Knowledge about Temporal Intervals*. Commun. ACM, Pages 832-843, 1983.

[2] D. ANICIC - *Event Processing and Stream Reasoning with ETALIS*. PhD thesis, Karlsruhe Institute of Technology, 2011.

[3] D. ANICIC, P. FODOR, S. RUDOLPH, R. STÜHMER, N. STOJANOVIC, R. STUDER - *A Rule-Based Language for Complex Event Processing and Reasoning.* Proceedings of the Fourth International Conference on Web Reasoning and Rule Systems (RR 2010), Pages 42-57. Springer, 2010.

[4] D. ANICIC, S. RUDOLPH, P. FODOR, N. STOJANOVIC - *Retractable Complex Event Processing and Stream Reasoning*. Proceedings of the 5th International Conference on Rule-Based Reasoning, Programming, and Applications, Pages 122-137. Springer, 2011.

[5] D. ANICIC, S. RUDOLPH, P. FODOR, N. STOJANOVIC - *Real-Time Complex Event Recognition and Reasoning – A Logic Programming Approach*. Applied Artificial Intelligence, 26(1-2):6-57, 2012.

[6] A. ARASU, S. BABU, J. WIDOM - *Cql: A Language for Continuous Queries over Streams and Relations.* International Workshop on Database Programming Languages, Pages 1-19. Springer, 2003.

[7] A. ARASU, S. BABU, J. WIDOM - *The cql Continuous Query Language: Semantic Foundations and Query Execution*. The VLDB Journal, 15(2):121-142, 2006.

[8] A. ARTIKIS, O. ETZION, Z. FELDMAN, F. FOURNIER - *Event Processing under Uncertainty*. Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, Pages 32-43. ACM, 2012.

[9] A. ARTIKIS, G. PALIOURAS - *Behaviour Recognition Using the Event Calculus*. Artificial Intelligence Applications and Innovations III, Pages 469-478. Springer, 2009.

[10] A. ARTIKIS, G. PALIOURAS, F. PORTET, A. SKARLATIDIS - *Logic-Based Representation, Reasoning and Machine Learning for Event Recognition.* Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems. ACM, 2010.

[11] A. ARTIKIS, M. SERGOT, G. PALIOURAS - *A Logic Programming Approach to Activity Recognition*. Proceedings of the 2nd ACM International Workshop on Events in Multimedia, Pages 3-8. ACM, 2010.

[12] A. ARTIKIS, M. SERGOT, G. PALIOURAS - *Run-Time Composite Event Recognition*. Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, Pages 69-80. ACM, 2012.

[13] A. ARTIKIS, A. SKARLATIDIS, G. PALIOURAS - *Behaviour Recognition from Video Content: A Logic Programming Approach*. International Journal on Artificial Intelligence Tools, 19(02):193-209, 2010.

[14] A. ARTIKIS, M. WEIDLICH, A. GAL, V. KALOGERAKI, D. GUNOPULOS - *Self-Adaptive Event Recognition for Intelligent Transport Management.* Big Data, 2013 IEEE International Conference on, Pages 319-325. IEEE, 2013.

[15] A. ARTIKIS, M. WEIDLICH, F. SCHNITZLER, I. BOUTSIS, T. LIEBIG, N. PIATKOWSKI, C. BOCKERMANN, K. MORIK, V. KALOGERAKI, J. MARECEK, *et al.* - *Heterogeneous Stream Processing and Crowdsourcing for Urban Traffic Management.* Proceedings of the 17th International Conference on Extending Database Technology (EDBT 2014). Athens, Greece, 2014.

[16] H. BECK, M. DAO-TRAN, T. EITER - *LARS: A Logic-Based Framework for Analytic Reasoning over Streams*. Artificial Intelligence, Pages 16-70, 2018.

[17] O. BERTRAND, P. CARLE, C. CHOPPY - *Chronicle Modelling Using Automata and Coloured Petri Nets.* 18th International Workshop on Principles of Diagnosis (DX-07), Pages 229-234, 2007.

[18] A. BOUFAIED, A. SUBIAS, M. COMBACAU - *Détection distribuée par reconnaissance floue de chroniques*. Journal Européen des Systèmes Automatisés, 40(2):233-259, 2006.

[19] P. CARLE, P. BENHAMOU, F.-X. DOLBEAU, M. ORNATO - *La reconnaissance d'intentions comme dynamique des organisations*. 6èmes Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA'98), 1998.

[20] P. CARLE, C. CHOPPY, R. KERVARC - *Behaviour Recognition using Chronicles*. Proc. 5th IEEE International Symposium on Theoretical Aspects of Software Engineering, Pages 100-107, 2011.

[21] P. CARLE, C. CHOPPY, R. KERVARC, A. PIEL - *Behavioural Analysis for Distributed Simulations*. 19th Asia-Pacific Software Engineering Conference (APSEC), 2012.

[22] P. CARLE, C. CHOPPY, R. KERVARC, A. PIEL - *Handling Breakdowns in Unmanned Aircraft Systems.* 18th International Symposium on Formal Methods (FM) - Doctoral Symposium, 2012.

[23] P. CARLE, C. CHOPPY, R. KERVARC, A. PIEL - *A Formal Coloured Petri Net Model for Hazard Detection in Large Event Flows.* 20th Asia-Pacific Software Engineering Conference (APSEC), 2013.

[24] P. CARLE, C. CHOPPY, R. KERVARC, A. PIEL - *Safety of Unmanned Aircraft Systems Facing Multiple Breakdowns*. 1st French Singaporean Workshop on Formal Methods and Applications (FSFMA), 2013.

[25] G. CARRAULT, M.-O. CORDIER, R. QUINIOU, F. WANG - *Temporal Abstraction and Inductive Logic Programming for Arrhythmia Recognition From Electrocardiograms.* Artificial Intelligence in Medicine, 28(3):231-263, 2003.

[26] Z. CHAOCHEN, C. A. R. HOARE, A. P. RAVN - *A Calculus of Durations*. Information Processing Letters, 40(5):269-276, 1991.

[27] L. CHITTARO, M. DOJAT - *Using a General Theory of Time and Change in Patient Monitoring: Experiment and Evaluation*. Computers in Biology and Medicine, 27(5):435-452, 1997.

[28] L. CHITTARO, A. MONTANARI - *Efficient Temporal Reasoning in the Cached Event Calculus.* Computational Intelligence, 12(3):359-382, 1996.

[29] S. CORADESCHI, T. VIDAL - *Accounting for Temporal Evolutions in Highly Reactive Decision-Making*. Fifth IEEE International Workshop on Temporal Representation and Reasoning, Pages 3-10, 1998.

[30] M.-O. CORDIER, C. DOUSSON - *Alarm Driven Monitoring Based on Chronicles.* 4th SafeProcess, Pages 286-291, 2000.

[31] M.-O. CORDIER, X. LE GUILLOU, S. ROBIN, L. ROZÉ, T. VIDAL - *Distributed Chronicles for On-line Diagnosis of Web Services*. 18th International Workshop on Principles of Diagnosis (DX-07), Pages 37-44, 2007.

[32] D. CRAM, B. MATHERN, A. MILLE - *A Complete Chronicle Discovery Approach: Application to Activity Analysis.* Expert Systems, 29(4):321-346, 2012.

[33] G. CUGOLA, A. MARGARA - *Processing Flows of Information: From Data Stream to Complex Event Processing.* ACM Computing Surveys (CSUR), 44(3):15, 2012.

[34] P. DOHERTY, G. GRANLUND, K. KUCHCINSKI, E. SANDEWALL, K. NORDBERG, E. SKARMAN, J. WIKLUND - *The Witas Unmanned Aerial Vehicle Project*. ECAI, Pages 747-755, 2000.

[35] P. DOHERTY, J. KVARNSTRÖM, F. HEINTZ - *A Temporal Logic-Based Planning and Execution Monitoring Framework for Unmanned Aircraft Systems.* 6th International Conference on Recent Advances in Intrusion Detection (RAID'03), 2009.

[36] P. DOHERTY, J. KVARNSTRÖM, F. HEINTZ - *A Temporal Logic-Based Planning and Execution Monitoring Framework for Unmanned Aircraft Systems*. Autonomous Agents and Multi-Agent Systems, Pages 332-377, 2009.

[37] M. DOJAT - *Realistic Model for Temporal Reasoning in Real-Time Patient Monitoring*. Applied Artificial Intelligence, 10(2):121-144, 1996.

[38] P. DOMINGOS, D. LOWD - *Markov Logic: An Interface Layer for Artificial Intelligence.* Synthesis Lectures on Artificial Intelligence and Machine Learning, 3(1):1-155, 2009.

[39] C. DOUSSON, P. GABORIT, M. GHALLAB - *Situation Recognition: Representation and Algorithms.* International Joint Conference on Artificial Intelligence (IJCAI), Pages 166-172, 1993.

[40] C. DOUSSON - *Extending and Unifying Chronicle Representation with Event Counters*. Proceedings of the 15th European Conference on Artificial Intelligence, ECAI'2002, Lyon, France, July 2002, Pages 257-261, 2002.

[41] C. DOUSSON, P. LE MAIGAT - *Chronicle Recognition Improvement Using Temporal Focusing and Hierarchization*. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Pages 324-329, 2007.

[42] T. VU DU'O'NG - *Découverte de chroniques à partir de journaux d'alarmes, application à la supervision de réseaux de télécommunications.* PhD thesis, Institut National Polytechnique de Toulouse, 2001.

[43] O. ETZION, P. NIBLETT - *Event Processing in Action*. Manning Publications Co., 2010.

[44] P. FODOR, D. ANICIC, S. RUDOLPH - *Results on Out-of-Order Event Processing.* Proceedings of the 13th International Conference on Practical Aspects of Declarative Languages, Pages 220-234. Springer, 2011.

[45] M. GHALLAB - *On Chronicles: Representation, On-Line Recognition and Learning.* KR, Pages 597-606, 1996.

[46] H.-E. GOUGAM, A. SUBIAS, Y. PENCOLÉ - *Timed Diagnosability Analysis Based on Chronicles.* 8th IFAC International Symposium on Fault Detection, Supervision and Safety of Technical Processes SAFEPROCESS'2012, Pages 1256-1261, 2012.

[47] F. HEINTZ - *Chronicle Recognition in the WITAS UAV Project, a Preliminary Report*. Swedish AI Society Workshop (SAIS2001), 2001.

[48] P. KAARELA, M. VARJOLA, L. P. J. J. NOLDUS, A. ARTIKIS - *Pronto: Support for Real-Time Decision Making*. Proceedings of the 5th ACM International Conference on Distributed Event-Based System, Pages 11-14. ACM, 2011.

[49] A. KIMMIG, B. DEMOEN, L. DE RAEDT, V. SANTOS COSTA, R. ROCHA - *On the Implementation of the Probabilistic Logic Programming Language Problog*. Theory and Practice of Logic Programming, 11(2-3):235-262, 2011.

[50] R. KOWALSKI, M. SERGOT - *A Logic-Based Calculus of Events*. New Generation Computing, 4(1):67-95, 1986.

[51] K. KUMAR, A. MUKERJEE - *Temporal Event Conceptualization*. Proceedings of the Tenth IJCAI Conference, 1987.

[52] X. LE GUILLOU, M.-O. CORDIER, S. ROBIN, L. ROZÉ, *et al.* - *Chronicles for On-Line Diagnosis of Distributed Systems*. Proceedings of the European Conference on Artificial Intelligence (ECAI), Pages 194–198, 2008.

[53] A. MÕHALLA, E. CRAYE, S. C. DUTILLEUL, M. BENREJEB - *Monitoring of a Milk Manufacturing Workshop Using Chronicle and Fault Tree Approaches*. Studies inInformatics and Control, 19(4):377-390, 2010.

[54] R. MILLER, M. SHANAHAN - *The Event Calculus in Classical Logic – Alternative Axiomatizations.* Electronic Transactions on Artificial Intelligence (http://www.etaij.org), 4, 1999.

[55] R. MILNE, C. NICOL, M. GHALLAB, L. TRAVE-MASSUYES, K. BOUSSON, C. DOUSSON, J. QUEVEDO, J.AGUILAR, A. GUASCH - *TIGER: Real-Time Situation Assessment of Dynamic Systems*. Intelligent Systems Engineering, 3(3):103-124, 1994.

[56] B. MORIN, H. DEBAR - *Correlation on Intrusion: An Application of Chronicles*. 6th International Conference on Recent Advances in Intrusion Detection (RAID'03), Pages 94-112. Springer, 2003.

[57] A. PASCHKE, M. BICHLER - *Knowledge Representation Concepts for Automated SLA Management.* Decision Support Systems, 46(1):187-205, 2008.

[58] A. PASCHKE, A. KOZLENKOV, H. BOLEY - *A Homogeneous Reaction Rule Language for Complex Event Processing*. Workshop on Event driven Architecture, Processing and Systems, 2007.

[59] Y. PENCOLÉ, A. SUBIAS - *A Chronicle-Based Diagnosability Approach for Discrete Timed-Event Systems: Application to Web-Services.* Journal of Universal Computer Science, 15(17):3246-3272, 2009.

[60] A. PIEL - *Reconnaissance de comportements complexes par traitement en ligne de flux d'évènements.* PhD thesis, Université Paris 13 / ONERA, 2014.

[61] F. PORTET - *Pilotage d'algorithmes pour la reconnaissance en ligne d'arythmies cardiaques*. PhD thesis, Université Rennes 1, 2005.

[62] R. QUINIOU, L. CALLENS, G. CARRAULT, M.-O. CORDIER, E. FROMONT, P. MABO, F. PORTET - *Intelligent Adaptive Monitoring for Cardiac Surveillance.* Computational Intelligence in Healthcare 4, Pages 329-346. Springer, 2010.

[63] R. RINCÉ, R. KERVARC, P. LERAY - *On the Use of WalkSAT-Based Algorithms for MLN Inference in Some Realistic Applications*. Proceedings of the 30th International Conference on Industrial, Engineering, and Other Applications of Applied Intelligent Systems, Pages 121-131, 2017.

[64] A. SKARLATIDIS, A. ARTIKIS, J. FILIPPOU, G. PALIOURAS - *A Probabilistic Logic Programming Event Calculus*. Journal of Theory and Practice of Logic Programming (TPLP), 15(2):213-245, 2015.

[65] A. SKARLATIDIS, G. PALIOURAS, G. A. VOUROS, A. ARTIKIS - *Probabilistic Event Calculus based on Markov Logic Networks.* 5th International Conference on Rule-Based Modeling and Computing on the Semantic Web, Pages 155-170. Springer, 2011.

[66] N. STOJANOVIC, A. ARTIKIS - *On Complex Event Processing for Real-Time Situational Awareness.* 5th International Conference on Rule-Based Reasoning, Programming, and Applications, Pages 114-121. Springer, 2011.

[67] A. SUBIAS, E. EXPOSITO, C. CHASSOT, L. TRAVE-MASSUYES, K. DRIRA - *Self-Adapting Strategies Guided by Diagnosis and Situation Assessment in Collaborative Communicating Systems.* 21st International Workshop on Principles of Diagnosis (DX 10), Pages 329-336, 2010.

[68] M. VARJOLA, J. LOFFLER - *Pronto: Event Recognition for Public Transport.* 17th ITS World Congress, 2010.

[69] K. WALZER, M. GROCH, T. BREDDIN - *Time to the Rescue - Supporting Temporal Reasoning in the Rete Algorithm for Complex Event Processing.* Database and Expert Systems Applications, Pages 635-642. Springer, 2008.

AUTHORS

Romain Kervarc graduated from the *École Normale Supérieure de Lyon* and obtained a Ph.D. in formal logic in 2007. Since then, he has been a full-time researcher at ONERA, where he is the head of research unit "Modelling and Engineering of Distributed Systems and Software". His research interests include system modelling, formal methods, complex event processing, temporal logic and mixed logical and stochastic approaches, with a particular focus on the application of such method to actual industrial systems.

Ariane Piel graduated from Université Paris 13 in 2014 where she received her Ph.D. in computer science. She also holds a M.Sc. in mathematical logic and foundations of computer science. From 2011 to 2016, she was a Ph.D. student and then a post-doctoral researcher at ONERA – The French Aerospace Lab, in the Department for System Design and Performance Evaluation. Since 2016, she has been a full-time research fellow at CEA-LIST. Her research interests are focused on logic and formal methods for the conception and evaluation of systems.