An Automata-based Monitoring Technique for Commitment-based Multi-Agent Systems*

Paola Spoletini[1] and Mario Verdicchio[2]

[1] Università dell'Insubria
via Ravasi 2, 21100 Varese, Italy
paola.spoletini@uninsubria.it
[2] Università degli studi di Bergamo
via Salvecchio 19, 24129 Bergamo, Italy
mario.verdicchio@unibg.it

Abstract. In open multi-agent systems (MASs) we cannot assume agents to be developed in a centralized fashion. Recent proposals of commitment-based communication frameworks aim at increasing such openness. Interaction with agents whose behavior does not follow a universal standard raises the need for some means of protection for each agent. In this work we propose an automata-based monitoring module that continuously supports an agent during its life in a MAS. Such module includes a *Word Composer* that observes exchanged messages and keeps track of significant past interactions to express an agent's input in the form of time-stamped words, and a *Word Analyzer* that processes such words and matches them against some properties expressed in linear temporal logic which are supposed to hold throughout the interactions.

1 Introduction

Communication may be considered as playing a fundamental role in increasing the openness of multi-agent systems (MASs), as interaction standards that need not take agents' internal architecture into account allow for systems populated by heterogeneous, independently developed entities.

The most significant agent communication language (ACL) standard proposed so far, FIPA ACL, despite some minor changes throughout the years, has always been providing mental-state-based specifications. Among the issues that rise from such an approach, the most compelling is probably the fact that programmers are supposed to create software with a specific architecture implementing such prescribed mental states.

To counter this limitation to MASs' openness, some researchers have proposed ACL standards with a commitment-based semantics [5, 15], according to

* This work is partially supported by the European Commission through the FP7 Projet PrimeLife and by the Department of Information Technology and Mathematical Methods of the University of Bergamo through the Young Researchers Mobility Fund.

which every communicative act is seen as the creation or the modification of a commitment binding the agent to the others. While mental states are subjective and private, commitments are objective and public, and can be stored in public records for further reference.

These advantages come with a cost, dealing with different aspects of a commitment based open MAS. Firstly, if agent interaction is expressed in terms of public commitments, every agent needs to check whether such commitments are fulfilled or not on the basis of the events that have occurred in the system. We might assume such task to be performed by a centralized service to which all agents subscribe, but as one of the commitment proposal's main aims is to increase openness in MASs, we might as well prescribe that each agent be provided with a monitoring module. Moreover, openness means also that an agent cannot rely on any assumption about the agents it interacts with other than a common communication framework.

This raises the need for a way to protect the agent from potentially harmful interactions which might end up with contradictory commitments (which would inevitably lead to a violation of one of them) or with commitments to actions that are not compatible with the agent's characteristics or resources. The agent's monitoring module, thus, would also help keep its interactions in an open system safe, by checking whether the events are compatible with the properties that are part of the agent's specifications.

We propose that such a module be implemented as a component comprised of two submodules: a *Word Composer* elaborates exchanged messages in the form of timestamped words, which are in turn processed by a *Word Analyzer* that, exploiting finite state automata on infinite words, analyzes the state of the MAS and checks whether some properties, expressed in linear temporal logic, hold.

We present our proposal in the remainder of this work, which is organized as follows. Section 2 provides the theoretical background about the concepts supporting commitment-based agent interactions and the monitoring module, which is illustrated in detail in Section 3; in Section 4 some of the most significant related work is referred to; finally, Section 5 concludes.

2 Background

Let us start with some linear temporal operators, their definitions, and some abbreviations, followed by a suitable content language to represent the commitment-based domain the agents are working in. Then, we illustrate the theoretical background on Büchi and alternating automata, which our monitoring module is based on.

2.1 LTL$^{\pm}$

The monitored properties are expressed using the notion of time as introduced by Linear Temporal Logic with both past and future modalities (LTL$^{\pm}$) [12]. In the following we describe a propositional version of LTL, supposing that the atomic

propositions belong to a specific Content Language (CL). LTL$^{\pm}$ is a modal logic in which modalities refer to time and, considered a set of atomic propositions CL, its syntax is given over it in BNF as follows:

$$\varphi ::= true|p|\neg\varphi|\varphi \vee \varphi| \mathsf{X}\varphi \ |\mathsf{G}^{+}\varphi \ |\mathsf{F}^{+}\varphi \ |\mathsf{Until}(\varphi,\varphi)$$
$$\mathsf{P}\varphi \ |\mathsf{G}^{-}\varphi \ |\mathsf{F}^{-}\varphi \ |\mathsf{Since}(\varphi,\varphi) \mid \mathsf{WUntil}(\varphi,\varphi) \mid \mathsf{Z}^{+}(\varphi,\varphi)$$

where the modal operators X, P, F^{+}, F^{-}, G^{+}, G^{-}, Until, Since, WUntil, Z^{+} are called *next*(time), *previous*(time), *eventually in the future*, *eventually in the past*, *always in the future*, *always in the past*, *until*, *since*, *weak until* and *until and no longer*, respectively. The boolean operators $\wedge$ and $\Rightarrow$ are obtained, as usual, by composing $\vee$ and $\neg$.

The semantics of LTL$^{\pm}$ is given on a Kripke structure M, that consists of a tuple $\langle S,\ R,\ L\rangle$, where S is a finte set of states, $R \subseteq S \times S$ is the transition relation, and $L : S \to 2^{CL}$ is a labeling function that labels each state with the propositions in CL that are true in that state. A path $\pi = s_0,\ s_1,\ \ldots$ is an infinite sequence of states in S such that, $\forall i \geq 0$, $(s_i, s_{i+1}) \in R$.

We give the semantics of LTL$^{\pm}$ formula on a Kripke structure M using the following notation. Let π^i in a path $\pi = s_0\ s_1, \ldots$ be the suffix of π that starts from s_i. If φ is a formula, $M,\ \pi \models \varphi$ means that φ holds in the state initial state s_0 of the path π in the Kripke structure M. The relation $\models$ is then defined inductively as follows:

- $M,\ \pi \models\ p$ iff $p \in L(s_0)$;
- $M,\ \pi \models\ \neg\varphi$ iff $M,\ \pi \not\models \varphi$;
- $M,\ \pi \models\ \varphi_1 \vee \varphi_2$ iff $M,\ \pi \models \varphi_1$ or $M,\ \pi \models \varphi_2$;
- $M,\ \pi \models \mathsf{X}\varphi$ iff $M,\ \pi^1 \models \varphi$;
- $M,\ \pi \models \mathsf{Until}(\varphi_1, \varphi_2)$ iff there exists $k > 0$ such that $M,\ \pi^k \models \varphi_2$ and, for all $0 \leq j < k$, $M,\ \pi^j \models \varphi_1$;
- $M,\ \pi \models \mathsf{P}\varphi$ iff there is a path π_* s. t. $\pi_*^1 = \pi$ and $M,\ \pi_* \models \varphi$;
- $M,\ \pi \models \mathsf{Since}(\varphi_1, \varphi_2)$ iff there is a path π_* s. t. $\pi_*^n = \pi$ and there is a $0 \leq k < n$ such that $M,\ \pi_*^k \models \varphi_2$ and, for all $0 \leq j < k$, $M,\ \pi_*^j \models \varphi_1$;

The rest of the temporal modalities can be expressed using the ones above as follows: $\mathsf{F}^{+}\varphi$=$\mathsf{Until}(true, \varphi)$, $\mathsf{G}^{+}\varphi$= $\neg\mathsf{F}^{+}\neg\varphi$, $\mathsf{F}^{-}\varphi$=$\mathsf{Since}(true, \varphi)$, $\mathsf{G}^{-}\varphi$= $\neg\mathsf{F}^{-}\neg\varphi$, $\mathsf{WUntil}(\varphi_1, \varphi_2)$=$\mathsf{G}^{+}\varphi_1 \vee \mathsf{Until}(\varphi_1, \varphi_2)$,
$\mathsf{Z}^{+}(\varphi_1, \varphi_2)$=$\mathsf{WUntil}(\varphi_1, \varphi_2) \wedge \mathsf{G}^{+}(\varphi_2 \Rightarrow \mathsf{G}^{+}\neg\varphi_1)$.

Notice that the operators X and P give also a quantitative notion of time, since they can identify temporal instants in the domain of natural numbers.

Given an integer $K > 1$, we allow also a more concise form to express the boolean combination of nested X (and P, respectively), in the following way:

- $\mathsf{X}^K\varphi$ ($\mathsf{P}^K\varphi$) stands for K nested $\mathsf{X}\varphi$ ($\mathsf{P}\varphi$) operators.
- $\mathsf{F}^{+}_{\bullet K}\varphi$ ($\mathsf{F}^{-}_{\bullet K}\varphi$) with $\bullet \in \{<, \leq\}$ stands for
  $\varphi \vee \mathsf{X}\varphi \vee \ldots \vee \mathsf{X}^{K-1}\varphi$ ($\varphi \vee \mathsf{P}\varphi \vee \ldots \vee \mathsf{P}^{K-1}\varphi$) or
  $\varphi \vee \mathsf{X}\varphi \vee \ldots \vee \mathsf{X}^{K}\varphi$ ($\varphi \vee \mathsf{P}\varphi \vee \ldots \vee \mathsf{P}^{K}\varphi$)
  for $\bullet = <$ and $\bullet = \leq$ respectively.

– $\mathsf{G}^{+}_{\bullet K}\varphi$ ($\mathsf{G}^{-}_{\bullet K}\varphi$) with $\bullet \in \{<, \leq\}$ stands for
$\varphi \wedge \mathsf{X}\varphi \wedge \ldots \wedge \mathsf{X}^{K-1}\varphi$ ($\varphi \wedge \mathsf{P}\varphi \wedge \ldots \wedge \mathsf{P}^{K-1}\varphi$) or
$\varphi \wedge \mathsf{X}\varphi \wedge \ldots \wedge \mathsf{X}^{K}\varphi$ ($\varphi \wedge \mathsf{P}\varphi \wedge \ldots \wedge \mathsf{P}^{K}\varphi$)
for $\bullet = <$ and $\bullet = \leq$ respectively .

Notice that we are working using natural numbers as domain and, in this scenario, past modalities do not add any expressive power with respect to classical LTL [11], but allow us to represent the required properties in a shorter and more elegant fashion.

The LTL$^{\pm}$ temporal operators may be considered as the domain-independent part of the language that allows for the description of the properties an agent needs to monitor, while the context of the MAS where the agent is running determines the domain the propositional atoms refer to. As our proposal deals with MASs with a commitment-based communication framework, let us briefly provide a language which is expressive enough to describe such type of interaction. In this perspective, a message exchange is viewed as an action performed by an agent to create or modify the commitments that bind it to other agents.

Events are reified, and each event token belongs to at least an event type. An event brought about by an agent is called an action, and we write $Done(e, x, \tau)$ to mean that event e of type τ is brought about by agent x. We use the "m-dash" character as a shorthand for existential quantification. For instance, $Done(e, -, \tau)$ is defined as $\exists x Done(e, x, \tau)$.

A commitment is a social state between agents comprised of four components: an *event* e that has created the commitment, a *debtor* x which is the agent who is committed, a *creditor* y which is the agent the debtor is committed to, and a *content* u which represents the state of affairs the debtor is committed to bring about, and the relevant predicate is $Comm(e, x, y, u)$. By including event e in the parameters of a commitment, we make it linkable to the event that generated it for furhter reference. As we do not intend to depart from classical first-order logic, we let the content of a commitment be represented by a term u, and we write $\lfloor u \rfloor$ to refer to the relevant LTL$^{\pm}$ formula.

Intuitively, a commitment is fulfilled when its content, or, more precisely, the LTL$^{\pm}$ formula corresponding to its content is true, and is violated when its content is false. Allowing for more expressive contents including commitments themselves would easily lead to situations in the likes of the "liar paradox", which are far from automatically manageable. Investigating the allowable extent of content language expressiveness lies beyond the scope of this work. In our view, an agent's monitoring module gathers the truth values of the propositional atoms included in a commitment's content $\lfloor u \rfloor$ at all the needed states, as prescribed by the temporal operators in $\lfloor u \rfloor$. As soon as the module is able to calculate a truth value for the whole formula, the agent knows whether the relevant commitment has been fulfilled or violated. This process is described in detail in the next section.

Agents create commitments by performing suitable tokens of *commitment manipulation* action types, like make commitment (mc). The reader may refer to [19] for further details about commitment manipulation. The effects of the

performance of a commitment manipulation action are illustrated in the form of axioms. The scope of the validity of formulae is limited to the class of LTL$^{\pm}$ models that fulfill the constraints imposed by such axioms. For instance, if an agent (not necessarily x or y) performs an action of making a commitment with x as debtor, y as creditor, and u as content, then the relevant commitment holds until it is either fulfilled, or violated, after which it no longer exists:

$$\mathit{Done}(e, -, \mathit{mc}(x, y, u)) \Rightarrow \mathit{Comm}(e, x, y, u)\mathsf{Z}^{+}\mathit{Fulf}(e, x, y, u) \vee \mathit{Viol}(e, x, y, u).$$

In many cases, a possible content language CL that allows for a significant description of a domain must exploit first-order logic's expressiveness. However, to be able to write the properties to be monitored in the form of LTL$^{\pm}$ formulae, we need to translate the first-order logic sentences into propositions. Current propositional encodings (naive propositionalizations) result in extremely large propositional encodings even for moderate applications. No more efficient solution has been found yet, even though some promising proposals can be found in the literature [13]. For our purposes in this work, we assume that the monitoring agent lives in a system where at each interaction a finite domain is in place, and all the relevant identifiers are agreed upon by the participating agents. This allows for the above-mentioned propositional encodings.

2.2 Finite Automata on Infinite Words

Let us remind here the definitions of classical and alternating Büchi automata (BAs [17] and AAs [4], respectively), which consitute the basis for our monitoring module. Formally, a BA A is a tuple $\langle \Sigma, S, s_0, \delta, F \rangle$, where: Σ is the finite set of the input symbols, S is a finite set of states, $s_0 \in S$ is the initial state, $\delta : S \times \Sigma \to 2^S$ is the transition relation, and $F \subseteq S$ is the set of accepting states. Differently from classical finite state automata on finite words, these automata will accept infinite words, using the Büchi condition, i.e., a word $w = a_0, a_1, \ldots$ (for each $i \geq 0$, $a_i \in \Sigma$) is accepted by a BA A if exists an infinite sequence $s = s_0 s_1 s_2 \ldots$ where, s_0 is the initial state, for each $i \geq 0$, $\delta(s_i, a_i) = s_{i+1}$ and at least one state $s \in F$ appears on π infinitely many times.

AAs can be defined as BAs with the only difference in the transition function that becomes $\delta : S \times \Sigma \to B^{+}(S)$, where $B^{+}(S)$ is the positive boolean combination of the elements in S, i.e., a boolean combination using $\wedge$ and $\vee$ but not $\neg$. These automata allow two modalities: nondeterminism, also called existential modality, and parallelism, also called universal modality. As seen in the definition above, the former, given an input letter, allows the automaton to fire the transition choosing where to move among different possible targets. A word is accepted if at least one of these alternatives generates an acceptance run. Symmetrically, the latter makes the automaton move with just one transition in more than one state. This can be seen as the creation of as many copies of the automaton as the states to reach with a universal branch. In this case a word is accepted if it is accepted by all the generated copies.

LTL only with future modality is strongly correlated to BA and AA [4]. In the following we will show how we exploit this correlation.

3 The Monitoring Module

Figure 1 provides an overview of the monitoring module we are proposing. It is comprised of two submodules: the Word Composer (WCS) and the Word Analyzer (WAS), which support an agent during its interactions in the MAS. The WCS includes a sniffing functionality to get a copy of every message that the agent exchanges. Among the sniffed messages, only those dealing with the atomic propositions that appear in the monitored property are selected. Not only the WCS processes the received data to prepare the input for the next component, but, should the monitored property contain past tense operators, the WCS is also in charge of keeping track of the relevant information for future evaluation of the truth value of formulae of this kind. Subsection 3.2 provides more details about the construction of the input in the form of a time-stamped word and the processing of past-directed temporal operators. Once the input is ready, it is sent to the WAS, which consists of an alternating automaton functioning as a language acceptor. An unaccepted WCS word means that the state of the MAS does not satisfy the property expressed by the monitored formula, and such violation is notified to the agent. In accordance with the criticality level of the task the agent is supposed to carry out, it will consider the notification from the WAS as a warning or it will abandon the MAS.

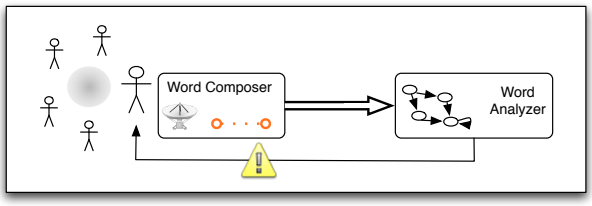

Fig. 1. The monitoring module.

3.1 Managing Temporal Aspects

LTL$^{\pm}$ allows for the creation of formulae with an arbitrary nesting of past and future tense operators. Nevertheless, the two temporal modalities can be processed separately. Gabbay [8] shows that a formula with nested operators can always be algorithmically broken down into subformulae only with future- and past-directed operators. This procedure is performed at a cost of a non-elementary blow up in the number of nested alternated modalities, and this complexity may result in a significant impact on the dimensions of the automaton needed to monitor the formula in worst case scenarios.

However, our approach mainly addresses formulae which are already separated or have a rather small number of nestings, as these are easier to relate with. Gabbay illustrates this technique by providing eight fundamental rules that are to deal with all the possible nesting combinations of the Since and Until operators, which are taken by the author as primitive. By exploiting the relations between these two operators and all the others, we can elaborate similar rules for our temporal language, which may be implemented as a formula preprocessor.

Moreover, as illustrated in more detail in Section 3.2, past operators that do not include any future operator can be processed into simple propositional atoms. In fact, at every instant, the past can be iteratively evaluated on the basis of the previous input, hence its truth value can always be available at the present time as the value of propositional letters. This means that we do not need to separate formulae in which past operators are embedded in future operators, but only need to take care of the opposite nesting (future operators nested in past operators).

Let us introduce an example of a simple formula to illustrate this approach: $\mathsf{G}^+(\mathsf{F}^-\mathsf{F}^+a \Rightarrow \mathsf{F}^+_{<10}b)$. In this formula the future tense operator F^+ is placed within the scope of the past tense operator F^-. By exploiting the well known equivalences $\mathsf{F}^+a \Leftrightarrow \mathsf{Until}(true,a)$ and $\mathsf{F}^-\ a \Leftrightarrow \mathsf{Since}(true,a)$ and Gabbay's rule

$$\begin{aligned}\mathsf{Since}(q,p\wedge\ \mathsf{Until}(B,A)) \Leftrightarrow\ & (\mathsf{Since}(q,p)\ \wedge\ \mathsf{Since}(B,p)\ \wedge\ B\ \wedge\ \mathsf{Until}(B,A))\ \vee\\ & (A\ \wedge\ \mathsf{Since}(B\wedge q,p))\ \vee\\ & (\mathsf{Since}(q,A\wedge q\wedge\ \mathsf{Since}(B,p)\ \wedge\ \mathsf{Since}(q,p))),\end{aligned}$$

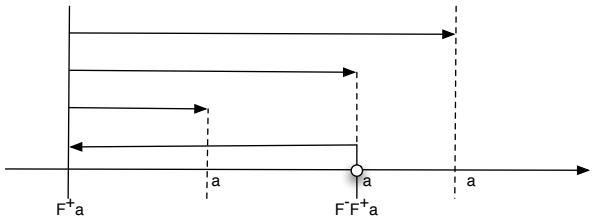

Fig. 2. Possible models for $\mathsf{F}^-\mathsf{F}^+a$.

By matching q, p, and B with $true$, and A with a, and by equivalences $\mathsf{Since}(true,true) \Leftrightarrow true$ and $true \wedge p \Leftrightarrow p$, we obtain

$$\mathsf{F}^-\mathsf{F}^+a \Leftrightarrow \mathsf{Until}(true,a)\ \vee\ a\ \vee\ \mathsf{Since}(true,a) \Leftrightarrow \mathsf{F}^+a\ \vee\ a\ \vee\ \mathsf{F}^-a.$$

More intuitively, $\mathsf{F}^-\mathsf{F}^+a$ holds at present time t_p if and only if there exists a $t < t_p$ where F^+a holds, which means that there must exist a $t' > t$ where a holds. Nothing is said about t' with respect to t_p, so that we could have $t' > t_p$, $t' = t_p$, or $t' < t_p$, which correspond to the three subformulae in the disjunction we obtained by Gabbay's rule. Some models for $\mathsf{F}^-\mathsf{F}^+a$ are depicted in Figure 2, while a model for the complete example is in Figure 3.

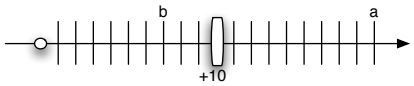

Fig. 3. A sequence of events that satisfies the example.

The outmost G^+ operator is not processed as part of the formula like the other operators, but taken as a procedural directive that prescribes that the monitoring task be continually performed.

3.2 The Word Composer

The input alphabet on which our monitoring module works is composed by propositions, which are either sniffed CL formulae translated into propositions or past subformulae evaluated on the basis of previously sniffed data and eventually flagged with a truth value. As our temporal model has a starting point, the input words are infinite on the right (in the future) but finite on the left (in the past), which means that a past subformula always relies on a finite support. These considerations have helped to prove the set of past-directed LTL$^{\pm}$ subformulae to be a language accepted by a deterministic BA[3] [16]. Our monitoring module exploits these results and relies on deterministic BAs to compute the truth value of past subformulae. In the following, we omit to represent these BAs and assume that the results of their computation is kept in a finite memory.

Let us focus on the unbounded operator $\mathsf{Since}(a1, a2)$, with $a1$ and $a2$ either present or past formulae (since we are working under the hypothesis of a separated form, we can ignore future-directed examples).The formula is true at a certain instant if in the past $a2$ was true and since then $a1$ has been true. In order to evaluate this formula at the present time, we would need to keep track of all the previous literals back to the first $a2$ or, even worse, back to the origin of the system. This problem can be overcome by evaluating Since at all instants, also if its truth value is not immediately required, for future evaluation in combination with other propositions describing the states of the system. More precisely, the truth value of $\mathsf{Since}(a1, a2)$ is evaluated using the algorithm represented by the flow diagram in Figure 4. An analogous function can be directly defined for G^- and F^- or, alternatively, these operators can be expressed in terms of Since.

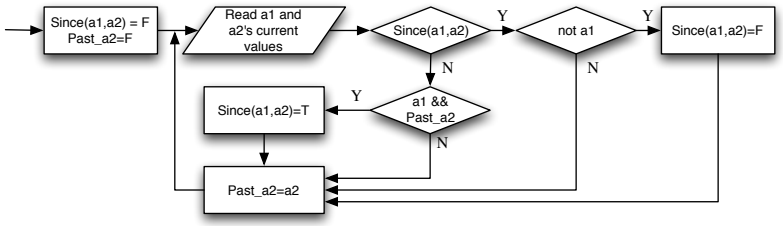

Fig. 4. Flow diagram of the algorithm to evaluate Since(a1, a2)

Bounded operators, i.e. P and the boolean combination of P, require a finite memory whose dimension is determined by the temporal constant (K) characterizing them. While with P we are just interested in the previous instant, with $\mathsf{F}^-_{\leq K}\varphi$ we need the last K values of φ in order to be able to evaluate the formula. Notice that if a bounded past operator is nested in another bounded past operator, its truth value needs as many memory bits as the sum of the two temporal constants in order to be computed.

[3] A deterministic BA is a BA such that the transition function is limited: $\delta : S \times \Sigma \rightarrow S$.

Taking these considerations into account, the example $\mathsf{G}^+((a \wedge \mathsf{F}^- c) \rightarrow \mathsf{F}^+_{<10} b)$ can be seen as $\mathsf{G}^+((a \wedge \texttt{evalF(c)}) \rightarrow \mathsf{F}^+_{<10} b)$, with `evalF(c)` computed using the algorithm represented in Figure 5.

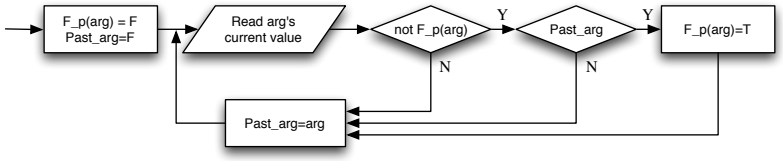

Fig. 5. Flow diagram of the algorithm to evaluate F^-(arg) (depicted as F_p)

3.3 The Word Analyzer

To illustrate how the MWA works, let us first focus on its input. The words composed as explained in Section 3.2 are timestamped, and the temporal distance between two adjacent literals (i.e. the numerical difference between the relevant timestamps) is not constant, but depends on when each data item has been sniffed. Our logic, on the contrary, is based on a model with a uniform distribution of time between successive states. To overcome this lack of temporal uniformity between the events that create the literals in the word and the flow of time in the system, we define an information preserving *filling procedure*. The basic idea is the following: under the hypothesis of choosing a small enough time unit[4], at each instant the atomic propositions are assigned the value they had the last time they were sniffed. Before the first sniffing, the propositions are assigned an initialization value.

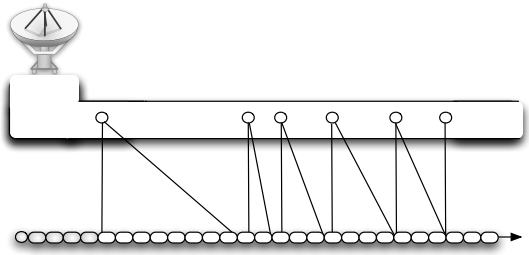

Fig. 6. An example of word filling

Figure 6 illustrates the filling procedure. The channel attached to the antenna carries the sniffed values of the propositions, which fill the word (the linear structure below the channel) until the successive sniffing. Under this hypothesis, our

[4] The time unit has to be small enough to guarantee no two changes of any atomic proposition values occur in the same time unit.

input words are coherent with the AA model presented in Section 2.2. As already mentioned, a strong correlation between AA and LTL exists which can be effectively exploited after the past subformulae elimination process (see Section 3.2) turns the monitored LTL$^{\pm}$ formula into an LTL sentence. A methodology to translate an LTL formula into an AA is presented in [18]. It consists in interpreting the subformulae with temporal operators as states, and the formula itself as the initial state, and in defining a particular transition function that preserves the composition relations between subformulae. We can exploit this approach for our monitoring purposes, but the following drawback must be taken into account. The concise operators built with a boolean combination of nested X operators lead to as many subformulae as the elements in the set we obtain from the transitive closure of the relation of being a subformula of the initial one. This may cause the construction of an AA with a large number of states, negatively impacting on the monitoring module's performance. To overcome this limitation, we follow the approach based on Alternating Modulo Counting Automata (AMCAs), AAs enriched with a finite set of finite counters, as proposed in [16].

An AMCA is a tuple $\langle \Sigma, S, s_0, \mu, \delta, F \rangle$, where: Σ is a finite set of the input symbols, S is a finite set of states, $s_0 \in S$ is the initial state, μ is a positive integer such that $Cnt = [0, \ldots, \mu]$ is a finite set of finite counters, $\delta : S \times \Sigma \times Cnt \rightarrow B^+(S \times Cnt)$ is the transition relation, $F \subseteq S$ is the set of accepting states. These automata do not add any expressive power to AAs, but they allow for a more concise representation of bounded operators. Thus, following the idea illustrated in [18], an LTL formula with metric operators can be translated into an AMCA.

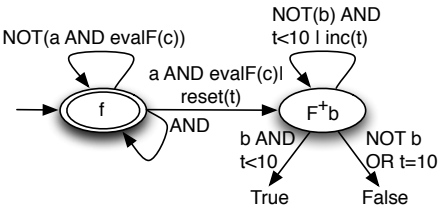

Fig. 7. The AMCA for the formula in the example. AND states for $\wedge$, OR for $\vee$, NOT for $\neg$ and f is the formula $\mathrm{G}^+((a \wedge \mathrm{evalF(c)}) \rightarrow \mathrm{F}^+_{<10} b)$.

To illustrate how an AMCA is generated from a formula, let us consider the example, once past operators have been transformed into propositional letters: $(\mathsf{G}^+((a \wedge \mathtt{evalF(c)}) \rightarrow \mathsf{F}^+_{<10} b))$. The resulting AMCA is in Figure 7. The automaton has four states: the initial state representing the formula, the state representing the temporal subformula $\mathsf{F}^+_{<10} b$ and True and False, which indicate termination in an acceptance and a non-acceptance state, respectively. The automaton cycles on the initial state as long as $a \wedge \mathtt{evalF(c)}$ is false. When this subformula becomes true, the automaton duplicates itself and a copy keeps on cycling on the initial state, while the other goes in $\mathsf{F}^+_{<10} b$ to check whether the consequent is satisfied. This second copy terminates after at most 10 time in-

stants in an acceptance state, if b is true by this deadline, in a non-acceptance state otherwise.

Notice that nothing prevents the system from generating another copy for $\mathsf{F}^+_{<10}b$ while one is still active. Indeed, the automaton will produce a copy every time $a \wedge \texttt{evalF(c)}$ is true. Since the copies in $\mathsf{F}^+_{<10}b$ will be active at most 10 time units, this means there may be as many as 10 active copies at the same time. This is very costly, but it could be much worse when unbounded operators F^+ , G^+ or Until are involved. For instance, with a slightly different formula $\mathsf{G}^+((a \wedge \texttt{evalF(c)}) \rightarrow \mathsf{F}^+b)$, as before, every time $a \wedge \texttt{evalF(c)}$ is true the automaton generates a copy to check the consequent, but, in this case, F^+b has no constraint on termination, so that infinitely many copies of the automaton may be created. This problem, which seems to seriously affect the feasibility of our approach, can be efficiently overcome without loss of expressive power. In many cases the duplication turns out to be unnecessary, especially in critical systems, where ending in a non-acceptance states is to trigger the agent's exit from the system. Exiting the system at the first failure actually guarantees that the maximum number of needed copies of the ACMA is $C = N + \Sigma_{i=0}^{M} K_i$, where N is the number of temporal subformulae, not including nested X operators, which are as many as M, with K_i being the relevant level of nesting. In other words, the monitoring module just needs one automaton for every temporal operator, except for X^{K_i}, for which K_i copies are required. Let us provide a more detailed account by analyzing each temporal operator.

- $\mathsf{F}^+\varphi$: when there is an active automaton waiting for φ to be true, a new copy would be useless, in that it would also wait for the same condition.
- $\mathsf{G}^+\varphi$: the existing automaton keeps on being active as long as φ is true, and this is exactly the same task that a new copy would perform.
- $\mathsf{Until}(\varphi, \psi)$: again, a second copy would have the same behavior of the existing one, in that a state with ψ true would satisfy both, a state with φ true and ψ false would keep them both active and waiting for the next instant to perform a new evaluation, and a state in which both propositions are false would lead both copies to a non-acceptance state.
- $\mathsf{X}^K\varphi$: this operator is punctual, in that, the relevant truth value is determined by a single state. Thus, if a new copy of the automaton is required, it is to evaluate a state which lies on the outside of the scope of the currently active automaton. No optimization is then possible. However, each copy will be active for exactly K time instants, which means that no more than K automata will be active at the same time.
- $\mathsf{G}^+_{\bullet K}\varphi$: let us suppose that in the situation depicted in Figure 8, with an automaton launched at instant 1, a new copy is required at instant 2. Automata 1 and 2 aim at checking whether φ is going to be true at all K instants of the intervals starting at 1 and 2, respectively. Resetting automaton 1's counter to zero at instant 2 is a way to achieve the same result without the need for the creation of automaton 2.
- $\mathsf{F}^+_{\bullet K}\varphi$: referring again to Figure 8, if at instant 2 automaton 1 is still active, it means that φ has not become true yet. A new copy created at instant 2 would

look for an occurrence of φ in interval $[2, 2+K]$. It should be noticed that if K does not become true by $1+K$, automaton 1 ends in a non-accepting state. Thus, the only interval that counts at instant 2 is $[2, 1+K]$, which means that automaton 1 suffices for the monitoring purposes.

Notice that the upper limit proposed for the number of copies of the automaton holds only for critical systems, where the agent is supposed to quit the monitoring process as soon as the first violation occurs. When the level of criticality is not an issue and the monitoring process is performed for statistical purposes, i.e., the monitoring module has to find all the violations and not only the first one, the number of needed copies becomes $C = U + \Sigma_{i=0}^{B} K_i$, where U is the number of unbounded subformulae and B is the number of bounded ones. The idea is that, since now we are interested in all the errors, when a bounded operator is involved, we want to know where the error occurs within the its scope, in order to associate the failure with the relevant event.

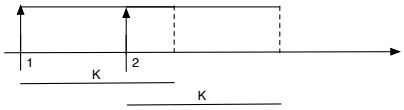

Fig. 8. Timeline to show the multiple copies.

3.4 An example

We experimented our monitoring technique on MASs using the JADE platform[5]. The agent of a MAS in JADE communicates using a message transport system, called *Agent Communication Channel*, that checks all the messages exchanged in the platform, including the messages from and to remote platforms. An extra module, called *sniffer*, is added to the MAS to analyze the exchanged messages. The sniffer has a cyclic behavior and it can intercept all the messages sent and received on the communication channel. In our case, it listens to all the messages from and to the monitored agent. Let us consider an example with an agent joining a MAS with a service provider. Before the agent's deployment, we can only verify the properties which deal solely with the agent's behavior. Once the agent joins the MAS, as we cannot make any assumption on the behavior of the other agents, we need to rely on the monitoring module to ensure that also more general properties involving all the entities in the MAS hold. For instance, we may be interested in the fact that all the commitments made by the service provider sp towards our agent x are fulfilled within 10 time units:

$$\mathsf{P}(\neg Comm(-, sp, x, u)) \wedge Comm(e, sp, x, u) \Rightarrow \mathsf{F}^{+}_{<10}(Fulf(e, sp, x, u)),$$

where $\lfloor u \rfloor = \phi$. Taking the definition of fulfillment into account, after the propositionalization of the formulae we obtain:

$$\mathsf{P}(\neg C_u) \wedge C_u \Rightarrow \mathsf{F}^{+}_{<10}(F_u) \text{ and } F_u \Leftrightarrow \phi,$$

[5] http://jade.tilab.com

that can be substituted by $\mathsf{P}(\neg C_u) \wedge C_u \Rightarrow \mathsf{F}^{+}_{<10}(\phi)$.

For the sake of simplicity we assume ϕ to be a propositional letter $a \in \mathrm{CL}$. As we have shown before, the automata for more complex formulae can be recursively built. The task then boils down to monitor $\mathsf{G}^{+}(\texttt{evalP}(\neg C_u) \wedge C_u \rightarrow \mathsf{F}^{+}_{<10} a)$ by means of the relevant automaton, as illustrated in the following pseudo-code. The first piece corresponds to the main program, comprised of an infinite loop that, every time a literal in the word built by the WCS is processed, it checks whether the propositions C_u and $\texttt{evalP}(\neg C_u)$ are true. If so, the main program calls for the monitoring of a by the function corresponding to the $\mathsf{F}^{+}_{<10}$ operator. Checking the condition not ActiveF guarantees that no unnecessary call will be performed. The flow diagram of the main is represented in Figure 9.

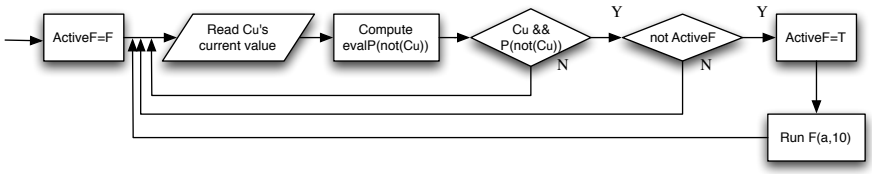

Fig. 9. Flow diagram of the main; 'Run' calls the relevant evaluation function.

The activated function takes the parameter of the call as the proposition to check, and it starts its run referring to the same literal in the word the main program is referring to. A local counter time is initialized to zero. The function is run at every new literal from the word, and it terminates when the relevant subformula can be evaluated to true or false.

The flow diagram of the evaluating function called with the Run command is depicted in Figure 10. It returns a success/failure value to the main (not depicted in Figure 9), which triggers the relevant signal and sets ActiveF to false to indicate that the evaluation function is no longer active.

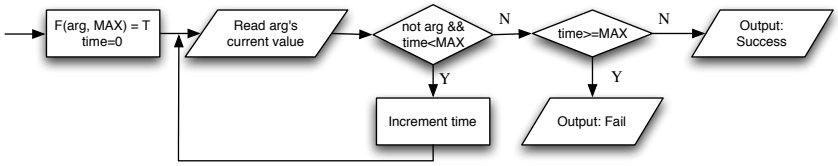

Fig. 10. Flow diagram of F

4 Related Work

Great interest in monitoring techniques arose in software engineering, and in particular in web services. Among the multitude of works in this field, particularly related to our work is Dynamo [3], a monitoring framework to assist the execution of workflow processes. Dynamo's synchronous integration of business and monitoring logics puts the execution of the business process on hold while the monitoring process checks the validity of a rule. Moreover, in [2] the authors

proposed a temporal extension to allow for the monitoring of more complex properties, both functional and non-functional. In this new version the authors define temporal operators exploiting the characteristics of their input language, hence, their semantics is defined on time-stamped words.

Dix et al. [7] propose adding a monitoring agent to a given MAS for debugging purposes. Given a planning problem, the monitoring agent generates all other possible plans to reach the same goal, then continuously checks and compares the messages exchanged by the other agents with all the plans. Should any incompatibility be detected, the monitoring agent generates an error file and reports to the MAS designer.

Guessoum et al. [9] regard monitoring as a process relying on a graph. Each node represents an agent in the MAS, and a weighted arc between two nodes stands for the communication load between the relevant agents. For each node there is a monitoring agent constantly updating the weights of the arcs the node belongs to, and notifying a supervisor agent whenever a significant change in a weight occurs. The supervisor thus has a general view on the communication in the MAS, and may prescribe the replication of some agents to avoid overload.

Kaminka et al. [10] propose a system which, following a non-intrusive approach, bases the monitoring process on overhearing of routine communication between agents which are members of a team aiming at the completion of a specific plan. The team is supposed to be geographically distributed, and the monitoring system sets off inference based on plan recognition against uncertainty due to non-perfect overhearing.

When restricting the context to *norms*, which can be seen as a specific deontic type of properties, several researchers, like Robles et al. [14] and Aldewereld et al. [1], propose to add a *governor* agent to the system for monitoring purposes. This approach requires much more structured interaction frameworks in the form of electronic institutions, to the detriment of openness and flexibility. As shown above, instead of one monitoring agent, we propose a simpler monitoring module within each interacting agent.

Cranefield [6] presents hyMITL$^{\pm}$, a rule language for specifying social expectations, and outlines an algorithm for rule compliance monitoring. hyMITL$^{\pm}$ relies on a branching model of time and is more expressive than the language we propose, which makes the monitoring process more complex a task. The current status of our work calls for a detailed comparison of the two approaches to establish the correct balance between expressiveness and efficiency.

5 Conclusions and Future Work

In this work we have proposed a monitoring module for analyzing the interaction of a single agent in a MAS. Our monitoring system is based on automata theory and takes advantage of research on model checking and web service monitoring systems. Our monitoring module contains two main components: a Word Composer, devoted to collect data from MAS communication and elaborate them in order to evaluate immediately present and past components, and a Word An-

alyzer, that analyzes such results to monitor the system during its evolution. There are still some interesting issues to tackle. First, we would like to enrich the logic used to state the properties, in order to define new classes of monitoring problems (e.g.: dense time, data aggregation functions), then, we aim at investigating the possibility to add a recovery mechanism, so that our monitoring module not only can detect errors, but it may also suggest possible ways to counter them.

References

1. H. Aldewereld, J. Vázquez-Salceda, F. Dignum, and J. J. Ch. Meyer. Verifying Norm Compliancy of Protocols. In *LNCS 3913*, pages 231–245. Springer, 2006.
2. L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. Validation of web service compositions. *IET Software*, 1(6):219–232, 2007.
3. L. Baresi and S. Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In *Proceedings of ICSOC'05*. ACM Press.
4. A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.
5. M. Colombetti. A Commitment–Based Approach to Agent Speech Acts and Conversations. In *Agents 2000*, pages 21–29, Barcelona, Spain, 2000.
6. S. Cranefield. Modelling and monitoring social expectations in multi-agent systems. In *LNCS 4386*, pages 308–321. Springer, 2007.
7. J. Dix, T. Eiter, M. Fink, A. Polleres, and Y. Zhang. Monitoring Agents using Declarative Planning. In *LNAI 2821*, pages 646–660. Springer, 2003.
8. D. M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *LNCS 398*, pages 409–448. Springer, 1989.
9. Z. Guessoum, M. Ziane, and N. Faci. Monitoring and organizational-level adaptation of multi-agent systems. In *Proceedings of AAMAS'04*, pages 514–521. ACM Press, 2004.
10. G. Kaminka, D. Pynadath, and M. Tambe. Monitoring teams by overhearing: A multi-agent plan recognition approach. *JAIR*, 17:83–135, 2002.
11. F. Laroussinie and P. Schnoebelen. A hierarchy of temporal logics with past. *Theoretical Computer Science*, 148(2):303–324, 1995.
12. M. Pradella, P. San Pietro, P. Spoletini, and A. Morzenti. Practical Model Checking of LTL with Past. In *Proceedings of ATVA 2003*, 2003.
13. D. Ramachandran and E. Amir. Compact propositional encodings of first-order theories. In *Proceedings of AAAI-05*, pages 340–345. AAAI Press, 2005.
14. A. Robles, P. Noriega, F. Cantú, and R. Morales-Menéndez. Enabling intelligent organizations: An electronic institutions approach for controlling and executing problem solving methods. In *LNCS 3789*, pages 275–286. Springer, 2005.
15. M. P. Singh. Agent Communication Languages: Rethinking the principles. *IEEE Computer*, 31:40–47, 1998.
16. P. Spoletini. *Verification of Temporal Logic Specification via Model Checking*. PhD thesis, Politecnico di Milano, Italy, 2005.
17. W. Thomas. Automata on Infinite Objects. *Handbook of Theoretical Computer Science (vol. B): Formal Models and Semantics*, pages 133–191, 1990.
18. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266, 1995.
19. M. Verdicchio and M. Colombetti. A Logical Model of Social Commitment for Agent Communication. In *LNCS vol.2922*. Springer, 2004.