Memory-driven dynamic behavior checking in Logical Agents

Stefania Costantini Arianna Tocchio

Università degli Studi di L'Aquila
Dipartimento di Informatica
Via Vetoio, Loc. Coppito, I-67010 L'Aquila - Italy
{stefcost, tocchio}@di.univaq.it

Abstract. In this paper, we deal with detecting behavioral anomalies in agents. In particular we consider agents defined via logic languages, and we take as a case-study the DALI language previously defined by the authors. We start proposing a formalization of aspects of agent memory and experience and we emphasize how the construction of memories can be a concrete aid for verification of future activities. The anomalies detection process is based on a set of constraints that use the agent past experience to identify behavioral traces different from those expected. Anomalies verification can be performed without stopping agent life and new constraints can be dynamically added thus allowing to check what the agent learns during its activity.

1 Introduction

Like human communities, agents can count only upon their capabilities for facing complex tasks and maintaining a high performance in dynamic environments where they are put at work. In order to cope with the unknown stimuli of the world, an agent needs to observe its past behavior and to deduce the best actions to do, trying to avoid the errors performed in previous situations. This motivates the importance of recording the most relevant facts which happened in the past and of recovering error and behavioral anomalies by means of appropriate strategies.

By providing agents with the capability of maintaining a track of past behavior and with a mechanism to elicit performed errors and a wrong behavior recovery strategies, it is possible to make these entities more powerful and reliable. The definition of frameworks for checking agent behavior during its life based on experience has not been really treated up to now. In fact, correspondingly to the agent framework complexity, there has been an increasing need for agent platforms whose entities would be capable of exhibiting a correct and rigorous behavior with respect to the expectations, but typically developers have applied model-checking techniques to system abstract models thus neglecting the verification of behavioral anomalies during the agent life.

The Model-checking paradigm allows to model a system S in terms of automata by building an implementation P_s of the system by means of a model-checker friendly language and then verifying some formal specifications. These are commonly expressed either as formulae of the branching time temporal logic CTL [4,23] or as formulae of

Linear Temporal Logic [14,30]. Model-checking techniques have been adopted in order to check systems implemented in AgentSpeak(L) [3]. In order to apply model-checking techniques, the authors have defined a variation of the language aimed at allowing its algorithmic verification. This variation is then submitted to model checkers. Penczek and Lomuscio have defined bounded semantics of CTLK [22], a combined logic of knowledge and time. The approach is to translate the system model and a formula ϕ, indicating the property to be verified, to sets of propositional formulae then submitted to a SAT-solver.

Methods for checking behavior are generally applied at the initial phase of the agent life. But, as it is well known, agents live in open environment where they can learn new knowledge or rules: then there are implicit difficulties for checking from time to time the behavior correctness by means of either model-checking or traditional approaches, that are generally static. Moreover, as mentioned before, model-checking techniques are applied by rewriting the interpreter in another language and this operation cannot be re-executed whenever the agent learns a new fact or rule.

In contrast to model checking, the deductive approach to verification uses a logical formula to describe all possible executions of the agent system and then attempts to prove the required property from this logical formula. The required properties are often captured by using modal and temporal logics. Deductive approaches have been adopted by Shapiro, Lesperance and Levesque that defined CASLve [28], a verification environment for the Cognitive Agent Specification Language. A limit of the theorem proving approach is the problems complexity, and thus a human interaction is often required.

Another possible approach to agent validation requires to observe the agent's behavior as it performs its task in a series of test scenarios before putting it at work. But this approach, as observed by Wallace in [31], by its very nature is incomplete, since all possible scenarios cannot be examined. Nor the future agent knowledge is knowable in advance. So, it is necessary to individuate a new mechanism capable of verifying the agent behavior correctness without stopping its life.

In this paper, we propose a method for checking the agent behavior correctness during the agent activity, based on maintaining information on its past behavior. This information is useful in that it records what has happened in the past to the agent (events perceived, conclusions reached and actions performed) and thus encodes relevant aspects of an agent's *experience*. If augmented by time-stamps, these records (that we call *past events*) constitute in a way the *history* of the agent activity. The set of past events evolves in time, and can be managed for instance by distinguishing the most recent versions of each past event, that contribute to the agent present perception of the state of the world. Past events can moreover be exploited for the purpose of self-checking agent activities: we propose in fact the introduction of specific constraints, aimed at checking if the entity has performed actions that it should not have done or has not performed actions that it should have done in a certain time and/or under some conditions. Alberti et al. in [1] have adopted a similar approach based on social constraints in order to model the interactions among (possibly heterogeneous) agents that form an open society. The main advantages of their approach are in the design of societies of agents, and in the possibility to detect undesirable behavior.

Our checking behavior proposal is not aimed at verifying the agents behaviors correctness in the context of communication acts exchange but to evaluate the reliability of a single agent by means of constraints. The constraints that we introduce are purposedly of a quite simple form so as to be easily and efficiently checked. As in this context we assume that our agent are not malicious, whenever the self-check should reveal an anomaly, the agent might for instance try a self-correction or report the anomaly to a supervisor agent. However, details of possible error-recovery strategies are outside the scope of this paper.

Another interesting class of techniques for agent behavior verification is based on variations of Kowalski and Sergots Event Calculus, used in conjunction with abduction. We intend in the future to perform a comparison between our behavioral constraints and these techniques. We mention here the interesting approaches in [27] and [2] that analyze safety properties and formalize Policy Specification. In [2], the abduction process is applied to a specification that models both the systems behavior and the policy specification, allowing to detect conflicts when the applicability of the policies is constrained on the runtime state of the system.

This paper is organized as follows. In Section 2 we discuss some kind of anomalies that agent behavior can reveal. In Section 3 we introduce concepts related to agent memory and experience, that we define more formally in Section 4. In Sections 5 and 6 we propose a set of constraints useful to detect behavioral anomalies. In Section 7 briefly discuss the semantics of our approach and, in Section 8, as a case-study, we show in detail the application of constraints in the DALI language [6] [7]. Finally, we conclude in Section 9.

2 Behavioral Anomalies Detection

Any mechanism for checking and possibly recovering agent behavioral anomalies relies on the ability to identify a model describing the expected agent behavior. This description can be provided at various levels of detail and under different points of view. However, any description should in general state what the agent has to do and what it must not do. In this context, we do not assume either a deep knowledge of the agent inner structure or the possibility of making all its state explicit. Rather, we assume to have a meta-description including a component specification of the expected observable agent behavior.

In order to define a framework for verifying the agent behavior correctness, we concentrate on six possible behavioral anomalies partially inspired by the work of Wallace in [31]:

- **Incorrect time execution:** An action or goal is accomplished at the incorrect time. This anomaly happens when we expect an agent to do an action or pursue a goal at a certain time but the action takes place before or after the established threshold.
- **Incorrect duration:** It is possible that an an action or goal lasts beyond a reasonable time or a specific related condition. In this case its duration is incorrect and determines an anomaly.

– **Omission:** The agent fails to perform the required action or to pursue its goal. This anomaly happens when an action/goal is not executed within a certain amount of time and cannot be executed later.
– **Intrusion:** The agent performs a goal or action that is not proper. This anomaly takes place when the expected behavior contains some actions/goals that the agent must not perform.
– **Duplication:** An action or a goal is repeated inappropriately. This happens when an agent performs more than once the same action/goal.
– **Incoherency:** An action or goal is executed a number of times greater than an expected threshold. The agent program can require that an action or a goal is executed a certain number of times but the agent performs that action beyond the prefixed threshold.

These behavioral anomalies can be very dangerous in critical contexts. Then, their capture not only improves the entities performance but prevents irreversible damages to the system. In order to identify similar anomalies, Wallace's approach [31] proposes a Behavior Comparison System based on traces. It can be viewed as a machine that takes two specifications of behavior as input and produces a summary of how these behaviors differ. We start from Wallace theoretical work, and develop a constraints-based system capable to detect the anomalies described above while the agent is active. This system, integrated in an agent framework, works for detection without stopping the agent life.

In the approach proposed in this paper, an agent is able to detect these anomalies via a self-check based on special constraint (clearly, in this context we assume that our agent are not malicious). What can be done if an anomaly is actually detected? Although details of possible error-recovery strategies are outside the scope of this paper, we can suggest that the agent might for instance try a self-correction or report the anomaly to a supervisor agent.

3 About agent experience

A rule-based agent consists of a knowledge base and of rules aimed at providing the entity with rational, reactive, pro-active and communication capabilities. The knowledge base constitutes the agent "memory" while rules define the agent behavior. We suppose that agents repeatedly execute a *observe-think-act cycle* as suggested by Kowalski in [16] in order to sense the environment, to decide the better action to perform and, finally, to modify the world by means of the action execution.

Imagine an agent that is capable of remembering the received external stimuli, the reasoning process adopted and the performed actions. Through "memory", the agent is potentially able to learn from experiences and ground what it knows through these experiences [18]. The interaction between the agent and the environment can play an important role in constructing its "memory" and may affect its future behavior. Most methods to design agent memorization mechanisms have been inspired by models of human memory as ([24],[21]).

In 1968, Atkinson and Shiffrin proposed a model of human memory which posited two distinct memory stores: short-term memory and long-term memory. This model

has been suggested by Gero and Liew for constructive memory whose implementation has been presented in [19]. Memory construction [in this model] occurs whenever a design agent uses past experiences in the current environment in a situated manner. In a constructive memory system, any information about the current design environment, the internal state of the agent and the interactions between the agent and the environment is used as cues in its memory construction process.

Gero and Liew introduce the notion of working memory, a workspace for reflective and reactive processes where explicit design-based reasoning occurs. Items of information within the working memory are combined with the stored knowledge and experiences, manipulated, interpreted and recombined to develop new knowledge, assist learning, form goals, and support interaction with the external environment. At this point it is clear that memory, experience and knowledge are strongly related. Correlation between these elements can be obtained via neural networks as in [19], via mathematical models as in [20] or via logical deduction.

The authors of this paper have proposed in [5], [6],[7] a method of correlating agent experience and knowledge by using a particular construct, the internal events, that has been introduced in the DALI language (though it can be in principle adopted in any computational logic setting). We have defined the "static" agent memory in a very simple way as composed of the original knowledge based augmented with *past events* that record the external stimuli perceived, the internal conclusions reached and the actions performed.

Past events can play a role in reaching internal conclusions. These conclusions, that are proactively pursued, take the role of "dynamic" memory that supports decision-making and actions: in fact, the agent can inspect its own state and its view of the present state of the world, so as to identify the better behavioral strategy in that moment. The agent re-elaboration of its experiences creates a particular view of the external world. By "particular" we mean that each agent, on the basis of its knowledge and experience, can interpret what has changed in the world in its peculiar manner. In our view therefore, an agent must record not only perceived external stimuli but also the internal conclusions reached by the entity and the actions performed. This allows in principle all aspects of agent behavior to be related, thus potentially improving its performance.

More specifically, *Past events* in our view, have at least two relevant roles:

- To describe the agent experience: for example, if an agent, after putting an apple on the table, takes it from the table and places it in the fruit-dish, all these actions must be recorded in the set of past events with the annotation of when they were performed, so that their sequence ca be reconstructed.
- To keep track of the state of the world and of its changes, possibly due to the agent intervention. In the above example, the agent should have a record (as past event) of the apple being on the table. Then, due to its own action (if successful), there will be a new record of the apple being in the fruit dish. The former item of information will be kept, but it will be no more "actual", while the latter one will now be "actual". Then, the most recent past events may be seen as representing the current "state of affairs".

With time, on the one hand past events can be overridden by more recent ones of the same kind (take for example temperature measurement: the last one is the "current" one) and on the other hand can also be overridden also by more recent ones of different kinds, which are somehow related.

In our approach, we introduce a set P of current "valid" past events that describe the state of the world as perceived by the agent. We also introduce a set PNV where we store all previous ones. The content of set PNV can be seen as the agent "memory" in a broader sense (we do not cope here with practical efficiency reasons that might force the agent to "purge" PNV in some way so as to regain store).

Then, we need a mechanism for keeping P up-to-date. The mechanism that we propose consists in defining in the agent program a set of constraints that express what and when to remove a past event from P.

4 Defining agent experience

We abstractly formalize an agent as the tuple $< Ag, P_{rg}, E, I, A >$ where Ag is the agent name, P_{rg} describes the agent behavioral rules, E is the external events set (events that the agent is capable to perceive and recognize), I is the internal events set (distinguished internal conclusions) and, finally, A is the actions set (actions that the agent can possibly perform). We emphasize that, while external events allow agents to react to the environmental stimuli, internal events generate the agent proactive capabilities. Finally, by using actions the agent influences the external world.

We suppose that each action performed and each external or internal stimulus received will be recorded by the agent in order to keep track of its past behavior. The events that happen are kept in a set $H = E \cup I \cup A$ where however each event in H is annotated (time-stamped) with the time when the event has happened. In practice, the set H is dynamically augmented with new events that happen, and is totally ordered w.r.t. the event time-stamps. Each event in H is then actually in the form $X : T_i$ where X is the event and T_i is its time-stamp; by abuse of notation for the sake of coinciseness we will occasionally indicate $X : T_i$ as X_i or simply X. We introduce a function $\Pi_Y^T(X)$ that takes in input an $X \in H$ and transforms it in the corresponding past form X_P^T where: T is the time-stamp of X; P is a postfix that syntactically indicates past events and Y is a label indicating what is X, i.e., if it belongs to E, I or A. I.e., $\Pi_Y^T(X) = X_P^T$

We can indicate the generation process of past events as the agent *history* h. The history is related to the order of set H, that reflects which events (non deterministically) happen. In fact, we assume to build the history according to this order. Several histories are thus in principle possible depending on both the interaction of the agent with the external environment and its internal choices.

$$h : init \xrightarrow{X_0 \in H} X_{P_0}^{T_0} \xrightarrow{X_1 \in H} X_P^{T_1} .... \xrightarrow{X_n \in H} X_P^{T_n}$$

while the set of past events P_h related to this specific history is:

$$P_h = \{X_{0P}^{T_0}, X_{1P}^{T_1}, ..., X_{nP}^{T_n}\}.$$

In the rest of this paper, we say P instead of P_h referring to the set of past events resulting from a generic agent history. Moreover, a generic element of P will be indicated by X_P^T or even X_P if the time-stamp is irrelevant. P synthesizes the whole agent life maintaining track of its actions, reactions and internal thoughts and its relevance is based on the possibility, for the agent, of knowing what it did and of deciding new strategies according to the old ones.

P is not static: it grows while the agent accomplishes its activities but it also loses those items that don't contribute any longer to the agent experience description. In the rest of this section, we introduce a set of constraints useful to determine which past events concur in the current agent life description and which don't. As mentioned before, past events no longer valid are eliminated from P and put in another set *PNV*, from which the agent can deduce some conclusions useful for performing its tasks.

Past Constraints define which past events must be eliminated and under which conditions. They are verified from time to time to maintain the agent memory consistent with the external world. More formally, we define a *Past Constraint* as follows:

Definition 1 (Past Constraint). *A Past Constraint has the syntax:*

$$X_{kP} : T_k, ..., X_{mP} : T_m \doteq X_{sP} : T_s, ..., X_{zP} : T_z, \{C_1, ..., C_n\}.$$

where $X_{kP} : T_k, ..., X_{mP} : T_m$ are the past events to be eliminated whenever past events $X_{sP} : T_s, ..., X_{zP} : T_z$ are in P and conditions $C_1, ..., C_n$ are true.

Each condition C_i can express either time or knowledge constraints. Consider the apple example introduced above. A corresponding constraint could be: $put\_apple\_in\_the\_table_P : T_1 \doteq put\_apple\_in\_the\_fruit\_dish_P : T_2, \{T_2 > T_1\}$.

The directive that it expresses is: if in P there is the past event $put\_apple\_in\_the\_fruit\_dish_P$ with a time greater than that of the past event $put\_apple\_in\_the\_table_P$, then delete from P the latter one (occurring in the head of the rule) because it is no longer valid and put it in the PNV set.

Formally, we can define a particular function, $\Phi(X)$ that determines the transmigration of the event X from P to PNV. This transmigration is based on the satisfaction of the related constraints. Let PC the set of Past Constraints. A generic past event X_P^T will be transformed into X_{PNV}^T iff there exists a Past Constraint in PC related to X_P^T whose conditions expressed in the body are satisfied.

$$\Phi(X_P^T) = \begin{cases} X_{PNV}^T & \exists C \in PC \mid C(X_P^T)\ is\ true \\ \\ X_P^T & else \end{cases}$$

More precisely, the PNV set will be composed of a certain number of X_{PNV}^T items:

$$PNV = \{X^{T_0}_{0PNV}, X^{T_1}_{1PNV}, ..., X^{T_n}_{nPNV}\}.$$

Past events in PNV may still have a relevant role for the entity decisional process. In fact, an agent could be interested for instance in knowing how often an action has been performed or a particular stimuli has been received by the environment. For this reason, we introduced some operators capable of deducing from PNV data useful to draw internal conclusions:

– **How often an action, a reaction or an internal conclusion has been performed.** This information can be obtained by the agent via the operator $\Delta^{PNV}_n(X)$ that returns the number n of occurrences of X in PNV. In fact, the parameter n has been adopted to indicate the number of occurrences that Δ has to return, and can take the special values *first* and *last* to indicate in fact that one is searching for either the first or the last occurrence, given the temporal order specified by time-stamps. For example, consider an agent that bought a car through instalments. If it wants to know when it paid all the money, it could check the number of instalments that has been paid:

  $$N = \#\Delta^{PNV}_n(pay\_instalment_A(Value, Date))$$

– **The first or last past event occurrence.** An agent would want to know when it performed an action, reached a conclusion or reacted to a certain event for either the first or the last time. This information can be obtained searching for event X in $P \cup PNV$:

  $$E_{first} = \Delta^{P \cup PNV}_{first}(X) \text{ or } E_{last} = \Delta^{P \cup PNV}_{last}(X)$$

  For example, the agent that bought the car could be interested in verifying when it paid the first or last instalment:

  $$First\_payment = \Delta^{P \cup PNV}_{first}(pay\_instalment(Value, Date))$$
  $$Last\_payment = \Delta^{P \cup PNV}_{last}(pay\_instalment(Value, Date))$$

– **Past event occurrences in a time interval.** Sometimes an agent is interested to know how many past event occurrences are present in a certain time interval T_1,T_2. In this case it will search the response in PNV:

  $$Occurrences\_list = \Delta^{PNV}_{T_1,T_2}(X)$$

  Suppose for instance that the bank that lent money to our agent, at a certain time contests the payment of the instalments from March to June. The agent is able to verify the situation via the following operator:

  $$Instalments = \Delta^{PNV}_{March,June}(pay\_instalment(Value, Date))$$

Finally, in order to formalize the consideration for which experience and knowledge concur to generate agent "memory", we formally define the *agent memory* M as:

$$M = \{X \in P \cup PNV \cup KB_{facts}\}$$

where KB_{facts} is the set of facts belonging to the agent knowledge base. Then, memory (in terms of facts) includes what the agent knows from start and what it records later. This "static" memory is then to be elaborated so as to reach internal conclusions about the state of either the world or the agent itself, that constitute the "dynamic" memory, or, in an alternative interpretation, the "self-consciousness" of the entity.

The agent *experience* synthesized by the sets P and PNV constitutes the starting point for our behavior checking approach.

5 Constraints for behavioral anomalies checking

Past events have a main role in defining our detection anomalies system. In fact, the possibility of discovering wrong behavioral sequences is strictly linked to the study of performed actions, reactions and internal conclusions. Given that past events resume previous agent behavior, as skillful archaeologists, we propose a method based on several *behavioral constraints* useful to discover specific traces. What kind of traces? In this section we cope with those suggested by the anomalies classification proposed in section 2. We introduce the following kinds of *behavioral constraints*.

– **Existential Constraint:** *Existential Constraints* contain a reference to the past, one to the present and a set of conditions useful to describe the class of histories that we are considering. They check whether the agent performed an action, a reaction or a conclusion that the expected behavior considers anomalous. More formally:

Definition 2 (Existential Constraint). *Let P be the set of past events, let $X_i \in P$, let T_i be its time-stamp and let $C_1, ..., C_n$ be a set of additional conditions (a similar reasoning can be performed with PNV events), for us an Existential Constraint has the following structure:*
$X_{kP} : T_k, ..., X_{mP} : T_m \exists \triangleleft X_{sP} : T_s, ..., X_{zP} : T_z, \{C_1, ..., C_n\}$.

The meaning is: if in P there are past events $X_{sP} : T_s, ..., X_{zP} : T_z$ and conditions $C_1, ..., C_n$ are true, then $X_{kP}^{\not\in} T_k, ..., X_{mP} : T_m$ past events must not be in P. Otherwise, we are in presence of an anomaly. The constraint is existential in that if just one of $X_{kP}^{\not\in} T_k, ..., X_{mP} : T_m$ is in P, this constitutes an anomaly.

– **Inquiring Constraint:** This kind of behavioral constraint checks past events that, considered some conditions, must be in P at a certain time. I.e., if an agent has performed some actions, reacted to some external stimuli or reached some internal conclusions and if some conditions are true, then necessarily, after a certain amount of time, the agent should have performed certain actions, reactions or reasoning as specified in the expected behavior. Or else, the entity is assuming an incomprehensible behavior. Formally, an *inquiring constraint* has the form:

Definition 3 (Inquiring Constraint). *An Inquiring Constraint has the structure:*

$X_{kP} : T_k, ..., X_{k+mP} : T_{k+m} \neg\exists \triangleleft X_{sP} : T_s, ..., X_{zP} : T_z, \{C_1, ..., C_n\}$.
where P is the set of past events, each $X_{iP} \in P$ with time-stamp T_i and $C_1, ..., C_n$ are a set of conditions.

The meaning is: if in P there are the past events $X_{sP} : T_s, ..., X_{zP} : T_z$ conditions and $C_1, ..., C_n$ are true, then we expect that at most at the time T_k the past event $X_{kP} : T_k$ be in P, and ... at most at time T_{k+m} the past event X_{k+mP} be in P. If any of them is lacking at the limit time, we are in presence of an anomaly.

In the next section we show how the behavioral constraints we have introduced allow us to detect the anomalies suggested by Wallace.

6 Detecting anomalies

In this section we propose, for each anomaly, how *behavioral constraints* are able to detect the specific traces. We will adopt some examples in order to explain the potentiality of our approach.

Incorrect time execution: *An action or goal is performed at the incorrect time*. Suppose for instance that our agent bought a goldfish and an aquarium. To keep the fish safe it is necessary that it first fills the aquarium up with water and then puts the animal inside. We will indicate an action with the postfix A for the sake of readability. So, the expected action sequence will be: $fill\_the\_aquarium_A$, $put\_inside\_fish_A$.

In order to check the actions execution correctness, we must verify that agent does not perform the second action before the first one. To this aim, we can adopt the *existential constraint*:

$fill\_the\_aquarium_P : T_1 \exists \triangleleft put\_inside\_fish_P : T_2, \{T_2 < T_1\}$.

indicating that if the agent has accomplished the action $put\_inside\_fish_A$ (it became past event) at the time T_2 and in P there is the past event $fill\_the\_aquarium_P^{T_1}$, which occurred later ($T_2 < T_1$) then the action sequence is not correct and we are in the presence of *Incorrect time execution* anomaly.

Incorrect duration: *An action or goal last beyond a reasonable time or a specifiied condition.* Consider the following simple program where we indicate external events (perceptions) with postfix E and we mean that opening the umbrella is a reaction to the perception of rain. We assume that an external event is recorded as a past events only after the related reaction has successfully taken place and that an action is recorded as a past events as soon as it has been successfully performed.

$rain_E : -open\_the\_umbrella_A$.

Then, in P we cannot have the past events $rain_P : T_1$ and $open\_the\_umbrella_P : T_2$ with $T_1 > T_2$. This would mean in fact that reaction has been considered to have

been successfully accomplished while the action had not yet been completed. It is an anomalous behavior. In order to detect this anomaly we can employ an *inquiring constraint*:

$open\_the\_umbrella_P : T_1, \neg\exists \triangleleft rain_P : T_2, \{T_1 < T_2\}$.

Omission: *The agent fails to perform the required action or pursue the required goal.* An action omission can be detected if we expect that the agent performs an actions sequence but one of the corresponding past events is lacking after a certain time limit. Suppose that our agent is on an island and must go fishing for surviving. For reaching its purpose, the agent must take the fishing-rod, then must bait the hook and finally must drop the fish-hook in the sea. So, the corresponding actions sequence will be: $take\_fishing\_rod_A$, $bait\_hook_A$, $drop\_fish\_hook_A$. Now, we can check an *omission anomaly* by adopting the following *inquiring constraint*:

$take\_fishing\_rod_P : T_k, \neg\exists \triangleleft$
$bait\_hook_P : T_2, drop\_fish\_hook_P : T_3, \{T_k < T_2 + Th_2, T_k < T_3 + Th_3\}$.

where Th_i are predefined time thresholds. The meaning is: if in P there are the past events $bait\_hook_P : T_2$ and $drop\_fish\_hook_P : T_3$ but at the time T_k defined by the conditions the past event $take\_fishing\_rod_P : T_k$ is absent, we can deduce that the corresponding action has been omitted.

Intrusion: *The agent performs a goal or action that is not allowed.* Suppose that our agent is situated in a critical environment where we need be sure that it never performs a dangerous action. We can verify if this happens by using an *existential constraint*. In particular, consider an agent that, if enters into the red room, must not push the green button. The corresponding constraint will be:

$push\_green\_button_P : T_1, \exists \triangleleft enter\_in\_red\_room_P : T_2, \{T_1 > T_2\}$.

Duplication: *An action or a goal is repeated inappropriately.* This anomaly can be detected by searching for two past events corresponding to the same action, reaction or conclusion having the same time T_i. The existential constraint capable of detecting this incorrect behavior is:

$push\_green\_button_P : T_1, \exists \triangleleft push\_green\_button_P : T_2, \{T_1 = T_2\}$.

Incoherency: *An action or goal is executed a number of times greater than an expected threshold.* This anomaly is strictly correlated with PNV events. In fact, the agent experience maintains the information on how often an action/goal has been accomplished. Consider again the agent that bought the car. It must pay twelve instalments in the current year. If it pays a further instalment, the expected behavior is violated:

$pay\_instalment(Value, Date))_P : T\ \exists \triangleleft$
$\{N = \Delta_n^{PNV}(pay\_instalment(Value, Date)), N > 20, T > 0\}$.

Behavioral constraints are to be checked from time to time by the system in order to point out the anomalies. When an incorrect behavior takes place, a particular past event can be generated. This event may contain information about the kind of violation for allowing the agent to activate possible recovery strategies. Moreover, by inspecting the sets P and PNV, one can discover the motivation of the entity anomalous behavior

examining all past events generated by the system. Consequently, appropriate countermeasures can be taken either by the agent itself, or by an external controller. In the next sections, after giving the semantics of the approach, we will put our idea at work in a real multi-agent system called DALI, a logic language capable of generating autonomous, reactive, pro-active and communicative agents.

7 Semantics of Past and Behavioral Constraints

The semantics of Computational Logic agent languages may in principle be expressed as outlined in [6] for the DALI language. In particular, given program P_{Ag}, the semantics is based on the following.

1. An *initialization step* where P_{Ag} is transformed into a corresponding program P_0 by means of some sort of knowledge compilation (which can be understood as a rewriting of the program in an intermediate language).
2. A sequence of evolution steps, where the reception of an event or a certain expired time threshold is understood as a transformation of P_i into P_{i+1}, where the transformation specifies how the event affects the agent program (e.g., it is recorded). The time threshold allows one to verify the behavioral constraints if no event is happened.

Then, one has a Program Evolution Sequence $PE = [P_0, ..., P_n]$ and a corresponding Semantic Evolution Sequence $[M_0, ..., M_n]$ where M_i is the semantic account of P_i (in [6] M_i is the model of P_i).

This semantic account can be adapted by transforming the initialization step into a more general knowledge compilation step, to be performed:

(i) At the initialization stage, as before.
(ii) When a new behavioral constraint is added.
(iii) In consequence to a violation evidence.

8 Anomalies detection in DALI

DALI [6] [7] [29] [9] [5] is an Active Logic Programming language designed in the line of [17] for executable specification of logical agents. The Horn-clause language is a subset of DALI, that in fact procedurally employs an Extended Resolution Procedure that interleaves different activities (based on the declarative semantics outlined above, and on an operational semantics based on Dialogue Games Theory [8]).

The reactive and proactive behavior of a DALI agent is triggered by several kinds of events: external, internal, present and past ones. All the events and actions are time-stamped, so as to record when they occurred. Past events represent the agent's "memory", that makes it capable to perform future activities while having experience of previous events, and of its own previous conclusions. Past events are kept for a certain default amount of time, that can be modified by the user through a suitable directive in the initialization file.

Definition 4 (Past Event). *A past event is syntactically indicated by the postfix P:*
$PastEvent ::= << Atom_P >>$

Original DALI program definition has been modified for introducing the set of behavioral constraints:

Definition 5 (DALI logic program). *A DALI logic program Pr_i is the tuple:*

$\langle Action_i, Reactive_i, Active_i, Hclause_i, Past\_constraints, Beh\_constraints_i \rangle$

where $Action_i$ is the set of the Action rules, $Reactive_i$ is the set of the Reactive rules, $Active_i$ is the set of the Active rules, $Hclause_i$ is the set of Horn clause rules, $Past\_constraints_i$ is the set of Past constraints and $Beh\_constraints_i$ is the set of Behavioral constraints. Reactive rules have an external or internal event in the head, Action rules specify the action-preconditions, Active rules have actions in their body, Horn clause rules are general prolog-like rules while Past and Behavioral constraints have been defined above.

Constraints on DALI agent behavior maintain a very similar syntax to that presented in Definitions 2 and 3.

Definition 6 (DALI Past constraints). *Let $E_{1P}, ..., E_{nP}$ be DALI past events, let $C_1, ..., C_s$ be conditions and $T_1, ..., T_n$ be time variables, we define a DALI past constraint as:*
$E_{1P} : T_1, ..., E_{kP} : T_k \ \sim / \ E_{k+1P} : T_{k+1}, ..., E_{nP} : T_n, \{C_1, ..., C_s\}.$

Definition 7 (DALI Behavioral constraints). *Let $E_{1P}, ..., E_{nP}$ be DALI past events, let $C_1, ..., C_s$ be conditions, let $T_1, ..., T_n$ be time variables and let Time indicate a precise moment, we define DALI Behavioral Constraints as follows:*

- **Existential constraint**
  $E_{1P} : T_1, ..., E_{kP} : T_k \ < / \ E_{k+1P} : T_{k+1}, ..., E_{nP} : T_n, \{C_1, ..., C_s\}.$
- **Inquiring constraint**
  $E_{1P} : at(Time) \ ?/ \ E_{k+1P} : T_{k+1}, ..., E_{nP} : T_n, \{C_1, ..., C_s\}.$

The meaning of these constraints corresponds to the Existential and Inquiring constraints introduced in previous sections. DALI constraints can be added to the logic agent program either at the initialization phase or when the agent is active. In fact, DALI agents are capable of learning rules and constraints through a particular mechanism, as discussed in [10]. This improvement allows one to expand the detection anomalies system potentialities by adapting the constraints to new knowledge learned by the entity during its life. Consider a simple DALI example on an agent being at home when it receives the perception that something dangerous is happened. In this example, we use the postfix E to indicate an external perception and A to identify the actions.

$danger_E :> once(ask\_for\_help)$.
$ask\_for\_help : -call\_police_A$.
$call\_police :< have\_a\_phone_P$.
$ask\_for\_help : -scream_A$.
$danger :< at\_home_P$.

The new tokens introduced have the following meaning: $:>$ denotes reaction, i.e. the body of the rule is involved whenever the head occurs as an external event; $:<$ denotes (for sake of clarity) that the rule expresses preconditions of an action. The meaning of this example is: if the agent receives from the environment the stimulus $danger_E$, it can react either by calling the police, if it has a phone, or by screaming. The reaction is allowed only if the agent is at home when the danger happens. The contextual information on the phone and the agent position is synthesized by *past events*. Two actions, $call\_police_A$ and $scream_A$ constitute agent alternative options and describe as past events a different evolution of the world. If the agent called the police, then it cannot have in its memory the past event of the opposite action $scream_P$. For keeping the world coherency we introduce in the agent program the following *past constraints*:

$call\_police_P : T \ \sim / \ scream_P : T_1, \{T < T_1\}$.
$scream_P : T \ \sim / \ call\_police_P : T_1, \{T < T_1\}$.

The directive expressed in the first constraint is: if in the agent memory there is the past event $call\_police_P$ happened at the time T and at the time T_1 greater than T the entity performs the action $scream_A$, the system must eliminate the previous action because it is no longer valid for describing the current world. The second constraint copes with the opposite situation.

At this point we complicate the agent behavior by introducing new rules:

$remain\_at\_home : -danger_P, call\_police_P, robber\_in\_kitchen_P$.
$remain\_at\_home_I :> go\_to\_bathroom_A, close\_the\_door_A$.
$go\_out : -danger_P, scream_P, robber\_in\_garage_P$.
$go\_out_I :> go\_to\_police\_station_A$.

We suppose that the agent, on the basis of performed actions $scream_A$ or $call\_police_A$, could draw some internal conclusions. In fact, when the agent decides to call the police it knows that the robber is in the kitchen. Then, not being able to escape, goes to the bathroom and locks the door. Instead, if the entity screamed and the robber is in the garage, then it goes to the police station. At this point, in order to guarantee the agent behavior correctness, we will introduce an *existential constraint* indicating that an agent cannot be in different places within a restricted time interval:

$go\_to\_police\_station_P : T \ < / \ remain\_at\_home_P : T_1, \{T_1 - 20 \leq T \leq T_1 + 20\}$.

By this constraint we define as anomalous behavior the situation in which the agent is within an interval of 20 seconds both in the police station and at home. In this case we have an incorrect execution.

Inquiring Constraints can help the agents system user to individuate actions that the agent has not performed but that it should have done within a certain time T. Suppose in our example that the police received the call and the policemen are sent to the agent's home. When they arrive, the agent receives the external stimulus $arrived\_police_E$ and it is happy because robber escapes:

$arrived\_police_E :> robber\_escapes_A.$
$i\_am\_happy : -robber\_escapes_P.$
$i\_am\_happy_I :> i\_embrace\_the\_policeman_A.$

However, we suppose that our agent will also be happy if it reaches the police station. Then we update the previous internal event adding a new condition:

$i\_am\_happy : -go\_to\_police_P.$

At this point, we have an agent that receives the external event $danger_E$ and, after a certain time it becomes happy because it meets the police. We can verify the agent behavior correctness by means of the following *Inquiring Constraint*:

$i\_am\_happy_{1P} : at([2006, 01, 16, 23, 44, 55])\ ?/$
$danger_P : T_1, at\_home_P : T_2, \{T_2 < T_1\}.$

The meaning is: after the dangerous situation described by the example, we will expect that, within a certain time interval fixed by the date, the entity will be happy. If this does not happen, the agent behavior is anomalous (Omission). We conclude this section emphasizing that Past and Behavioral Constraints are attempted by the DALI interpreter from time to time in order to detect as quickly as possible the anomalies.

9 Concluding Remarks

In this paper we have presented an approach to update agent memory and to detect behavioral anomalies by using logic constraints. The approach is based on introducing particular events, past events, that records what has happened. Past events are used to identify contexts in which agents adopt a behavior different from the one expected. Relating past behavior to future one is also proper of Temporal Logic. This is a special branch of modal logic that investigates the notion of time and order. With Temporal Logic one can specify and verify how components, protocols, and objects behave as time progresses ([11],[26]). Temporal Logic is also the core of an important language for multi-agent systems, Concurrent METATEM [12]. Despite Concurrent METATEM demonstrated that Temporal Logic is a fruitful land to generate software entities, some perplexities on its use in time-critical applications remain, due to limited efficiency.

Our approach aims in principle at introducing mechanisms similar to that of Temporal Logic but based on a simple and efficient constraint language. We are conscious that detecting behavioral anomalies is not sufficient in multi-agent systems. Actually, one should provide not only a mechanism to point out anomalies during the agent life but also error recovery strategies usable by the running agent without asking for human intervention. This is in fact a topic of our future research.

References

1. M. Alberti, M. Gavanelli, E. Lamma, P, Mello and P, Torroni, *An Abductive Interpretation for Open Agent Societies*. AI*IA 2003: 287-299
2. A. K. Bandara, E. C. Lupu and A. Russo. Using Event Calculus to Formalise Policy Specification and Analysis. In Proceedings of the 4th IEEE international Workshop on Policies For Distributed Systems and Networks (June 04 - 06, 2003). POLICY. IEEE Computer Society, Washington, DC, 26.

3. R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge, *Model checking agentspeak*. In Proceedings of the Second international Joint Conference on Autonomous Agents and Multiagent Systems (Melbourne, Australia, July 14 - 18, 2003),2003. AAMAS '03. ACM Press, New York, NY, 409-416. DOI= http://doi.acm.org/10.1145/860575.860641
4. E. M. Clarke, O. Grumberg and D. A. Peled, *Model Checking*, The MIT Press: Cambridge,MA,2000.
5. S. Costantini, *Towards active logic programming*, In A. Brogi and P. Hill, editors, *Proc. of 2nd International Workshop on component-based Software Development in Computational Logic (COCL'99)*, PLI'99, (held in Paris, France, September 1999), Available on-line, URL http://www.di.unipi.it/
brogi/ResearchActivity/COCL99/proceedings/index.html.
6. S. Costantini and A. Tocchio, *A Logic Programming Language for Multi-agent Systems*. In S. Flesca, S. Greco, N. Leone, G. Ianni (eds.), Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002, LNAI 2424, Springer-Verlag, 2002.
7. S. Costantini and A. Tocchio, *The DALI Logic Programming Agent-Oriented Language*. In J. J. Alferese and J. Leite(eds.), Logics in Artificial Intelligence, Proceedings of the 9th European Conference, Jelia 2004, Lisbon, September 2004. LNAI 3229, Springer-Verlag, Germany, 2004.
8. S. Costantini, A. Tocchio and A. Verticchio, *A Game-Theoretic Operational Semantics for the DALI Communication Architecture*, proc. of WOA04, Turin, Italy, December 2004, ISBN 88-371-1533-4.
9. S. Costantini and A. Tocchio, *About declarative semantics of logic-based agent languages*. In: "Declarative Agent Languages and Technologies", (revised selected papers presented at DALT 2005), Lecture Notes in Artificial Intelligence, Springer-Verlag, Germany, to appear.
10. , S. Costantini and A. Tocchio, *Learning by Knowledge Exchange in Logical Agents*. In: Simulazione ed Analisi Formale di Sistemi Complessi, Proceedings of WOA05, Universit di Camerino, Novembre 2005.
11. D. Drusinsky, *Monitoring Temporal Rules Combined with Time Series*. In Proc. of CAV'03: Computer Aided Verification, volume 2725 of Lecture Notes in Computer Science, pages 114–118, Boulder, Colorado, USA, 2003. SpringerVerlag.
12. M. Fisher, *Concurrent METATEM - A Language for Modeling Reactive Systems*. In Parallel Architectures and Languages, Europe (PARLE), Munich, Germany, June 1993. SpringerVerlag.
13. M. Fisher and M. Wooldridge, *Executable Temporal Logic for Distributed A.I.*. In Proc. of the 12th International Workshop on Distributed Artificial Intelligence, Hidden Valley, PA, May 1993. http://citeseer.ist.psu.edu/article/fisher93executable.html
14. G. Holzmann, *Design and Validation of Computer Protocols*. Prentice Hall International: Hemel Hempstead, england, 1991.
15. M. Kacprzak, A. Lomuscio and W. Penczek, *Verification of Multiagent Systems via Unbounded Model Checking*. In Proc. of the Third international Joint Conf. on Autonomous Agents and Multiagent Systems - Volume 2 (New York, New York, July 19 - 23, 2004), pp. 638-645. DOI= http://dx.doi.org/10.1109/AAMAS.2004.296
16. R. Kowalski and F. Sadri, *Logic Programming and the Real World* . Logic Programming Newsletter, 2001.
17. R. A. Kowalski, *How to be Artificially Intelligent - the Logical Way*. Draft, revised February 2004, Available on line, URL
http://www-lp.doc.ic.ac.uk/UserPages/staff/rak/rak.html.
18. P-K, Liew and JS, Gero, *A memory system for a situated design agent based on constructive memory*. In Eshaq, A., Khong, C., Neo, K., Neo, M., and Ahmad, S. (eds.), Proc. of CAADRIA2002, Prentice Hall, New York, pp. 199-206.

19. P-K, Liew and JS, Gero, *An Implementation model of constructive memory for a design agent*. In Agents in Design 2002, J. S. Gero and F. Brazier (eds.), 257-276. University of Sydney, Australia: Key Centre of Design Computing and Cognition.
20. K. Lerman, and A. Galstyan, *Agent memory and adaptation in multi-agent systems*. In Proc. of the Second International Joint Conf. on Autonomous Agents and Multi-agent Systems (Melbourne, Australia, July 14 - 18, 2003) AAMAS '03. ACM Press, New York, NY, 797-803. DOI= http://doi.acm.org/10.1145/860575.860703
21. R. H. Logie, *Visuo-Spatial Working Memory*. Lawrence Erlbaum: Hove, 1995.
22. A. Lomuscio, T. Lasica, and W. Penczek. *Bounded model checking for interpreted systems: preliminary experimental results*. In M. Hinchey (ed.), Proc. of FAABS II, LNCS 2699. Springer Verlag, 2003.
23. K. L. McMillan, *Symbolic Model Checking*. Kluwer Academic Publishers: Boston, MA, 1993.
24. D.G. Pearson and R. H. Logie, *Working memory and mental synthesis*. In S. O'Nuallan (Ed.), Spatial Cognition: Foundations and applications. John Benjamins Publishing Company.
25. D.G. Pearson, A. Alexander, and R. Webster, *Working memory and expertise differences in design*. In J. Gero, B. Tversky, and T. Purcell (eds.), Visual and Spatial Reasoning in Design II. Sydney: Key Centre of Design Computing and Cognition.
26. A. Pnueli, *The Temporal Logic of Programs*. In: Proc. 18th IEEE Symp. on Foundations of Computer Science, pp. 46-57, 1977.
27. A.Russo, R.Miller, B. Nuseibeh, and , J. Kramer. An Abductive Approach for Analysing Event-Based Requirements Specifications. In Proc. of the 18th international Conf. on Logic Programming (July 29 - August 01, 2002). P. J. Stuckey, Ed. Lecture Notes In Computer Science, vol. 2401. Springer-Verlag, London, 22-37.
28. S. Shapiro, Y. Lesprance, and H. J. Levesque, *The cognitive agents specification language and verification environment for multiagent systems*. In Proceedings of the First international Joint Conference on Autonomous Agents and Multiagent Systems: Part 1 (Bologna, Italy, July 15 - 19, 2002). AAMAS '02. ACM Press, New York, NY, 19-26.
29. A. Tocchio. *Multi-Agent sistems in computational logic*. Ph.D. Thesis, Dipartimento di Informatica, Universitá degli Studi di L'Aquila, 2005.
30. M. Y. Vardi, *Branching vs. linear time: Final showdown*. In T. Margaria and W. Yi (eds.), Proc. of the 2001 Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001 (LNCS Volume 2031), pages 1-22. Springer-Verlag: Berlin, Germany, april 2001.
31. S. A. Wallace, *Identifying Incorrect Behavior: The Impact of Behavior Models on Detectable Error Manifestations*. Fourteenth Conf. on Behavior Representation in Modeling and Simulation (BRIMS-05) 2005.