On Coordination, Autonomy and Time

(Extended Abstract)

Lăcrămioara Aştefănoaei
L.Astefanoaei@cwi.nl

Frank S. de Boer
F.S.de.Boer@cwi.nl

Mehdi Dastani
mehdi@cs.uu.nl

ABSTRACT

Timed coordination is an important issue in the developement of multi-agent systems. However, introducing a formalism for modelling time and achieving coordination while respecting the autonomy of the agents is still a challenge. This paper describes a formalism for a timed coordination mechanism which conciliates the "free thinking" of an agent.

Categories and Subject Descriptors: I.6.5 Model Development: Modelling methodologies

General Terms: Languages, Theory, Verification.

Keywords: Multi-Agent Systems, Timed Coordination, Autonomy.

1. MOTIVATION

Considering that the behaviour of a multi-agent system is simply the sum of the behaviours of individual agents is a too unrealistic idea since interaction is ignored. This is one of the reasons for introducing message passing or channel based communication. However, communication is "one" way to interact. Another possibility arises when one introduces coordination patterns. There are different paradigms to achieve this, for instance, using coordination languages like Reo [2]. Or, at a higher level of specification, there are logics for normed based coordination [5], [4]. In these approaches, further questions arise, e.g., how to implement normative concepts? Do coordination and normative languages have the same expressivity power? Should they be used in combination? Furthermore, modelling time in multi-agent systems is also of concern since it allows one to model deadlines, timeouts, action schedulling or dynamic behaviour. Is there a general design methodology to model time and implement coordination artifacts?

It is this latter question we approach. We take as a starting point BUpL [3] as an agent implementation language and we propose an *action-based coordination* mechanism which restricts the execution in time of BUpL agents by means of global synchronisation and ordering conditions. We refer to such mechanisms as *timed choreographies*. We find choreographies as a natural ingredient in scenarios where synchronisation is important, take, for example, two agents being assigned the task of lifting a table together. Furthermore, the advantage of the infrastructures we propose lies in their *exogenous* feature: the maintenance of the agent's internals is separated from the coordination pattern. We stress that choreographies are oblivious to the nature of actions and they control without having to know the internal structure of the agent. Thus the *autonomy* of an agent is preserved since nobody can change its internals but itself.

In order to model timed choreographies we adapt the theory of timed automata [1]. Having *timed choreographies*, however, does not make much sense without having time in the agents' programs. Thus, in our case, BUpL needs to be extended. In this respect, we have in mind that basic actions are a *common ontology* shared by all agents. Since the nature of basic actions does not specify *when* to be executed, our extension is thought such that the ontology remains timeless and "when" becomes a specific part of the syntax of the language.

We emphasise that all the effort of introducing the formalism of timed choreographies in multi-agent systems is motivated by the need to perform verification. Multi-agent systems are clearly more complex structures, and their verification tends to become harder. It is our intention to focus on the correcteness of a multi-agent system implementation with respect to a multi-agent system specification and a timed choreography. We achieve this by extending the refinement relation defined in [3] as action trace inclusion. In this way, we obtain a modular framework where it is enough to verify individual agents and/or the timed choreography in order to conclude properties about the whole system. Though the idea is quite simple, there are still some technicalities that need to be fixed. Traditionally, the operational semantics of a multi-agent system is given in terms of sets of traces of actions. However, choreographies might introduce deadlocks and since traces are a too coarse notion and do not reflect deadlocks we need to reconsider it. From the same reason we can no longer consider refinement being defined as trace inclusion, but look further for a finer notion of traces which allows us to consider that timed choreographies preserve the absence of deadlock.

2. APPROACH

We describe our methodology by means of an example. We imagine a scenario where a table can be lifted only by two agents. In order to complete their task, the agents should synchronise their actions. Furthermore, they are supposed to start their task within 5 time units and should complete it in less than 1 time unit. We achieve this by implement-

Cite as: On Coordination, Autonomy and Time, (Extended Abstract), Lăcrămioara Aştefănoaei, Frank S. de Boer, Mehdi Dastani, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 1357–1358

ing a choreography as the timed automaton illustrated in Figure 1. A *timed automaton* is a finite transition system extended with real-valued clock variables. Time advances only in states since transitions are instantaneous. Clocks can be reset at zero simultaneously with any transition. At any instant, the reading of a clock equals the time elapsed since the last time it was reset. States and transitions have *clock constraints*, defined by the following grammar:

$$\phi_c ::= x \leq t \mid t \leq x \mid x < t \mid t < x \mid \phi_c \wedge \phi_c,$$

where $t \in \mathbb{Q}$ and x is a clock. What is specific to timed

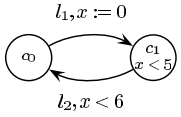

Figure 1: A Timed Choreography

choreographies is the labelling. We use labels of the form $\|_{j=1}^{n}\ (i_j, a_j)$, to specify that the agents $i_1, \ldots, i_n$ need to synchronise their actions $a_1, \ldots, a_n$. For example, in Figure 1 l_1 is $\|_{j=1}^{2}\ (i_j, goto(table))$ and l_2 is $\|_{j=1}^{2}\ (i_j, lift(table))$.

We take BUpL as the agent implementation language. The configuration of a BUpL agent is a tuple $\langle \mathcal{B}_0, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where $\mathcal{B}_0$ is an initial belief base, $\mathcal{A}$ is a set of basic actions, $\mathcal{P}$ is a set of plans and $\mathcal{R}$ is a set of repair rules. Beliefs are ground atoms, and basic actions are pairs of pre- and post-conditions. The mechanism of basic actions is based on an update operation. When the pre-condition of the action holds, beliefs are added and/or removed from the belief base, according to the post-condition. Basic actions are understood as a common ontology which can be shared among different agents. Plans are orderings ";" and/or choices "+" of actions. Repair rules are applied when actions fail and their purpose is to replace the plan in execution. Figure 2 presents a BUpL agent which has the choice of either going to the table or lifting it. The repair rule makes the agent execute p_0 in the case it didn't succeed in lifting the table.

Time is application specific and thus we need to extend

$$\begin{array}{lcl} \mathcal{B}_0 & = & \{\ at(table)\ \} \\ \mathcal{A} & = & \{\ goto(X) = (\ \neg at(X),\ at(X)\ ), \\ & & \quad lift(X) = (\ \neg lifted(X),\ lifted(X)\ )\ \} \\ \mathcal{P} & = & \{\ p_0 = (goto(table) + lift(table))\ \} \\ \mathcal{R} & = & \{\ \neg\ lifted(table) \leftarrow p_0\ \} \end{array}$$

Figure 2: A BUpL Toy Agent

BUpL with time constructions. For this, we consider that agents have a set of local clocks and that clock valuations are recorded in their mental states. We now pose the problem of how agents make use of clocks. We bare in mind the design principle: "the specification of basic actions does not come with time", thus actions are instantaneous. This implies that, in order to make the time pass and to keep intact the ontology of basic actions (basic actions being specified only in terms of pre/post conditions) we need to introduce a special type of actions. We refer to such actions as *delay actions*, $\phi \rightarrow I$, where ϕ is a query on the belief base and I is a clock constraint like $x \leq 1$. Basically, their purpose is to make time elapse in a mental state where certain beliefs hold. We further extend BUpL plans such that previous calls $a; p$ are replaced by $(\phi_c, a, \lambda); p$ and $(\phi \rightarrow I); p$, where ϕ_c is time constraining the execution of action a and λ is the set of clocks to be reset. For example, a timed version of the plan p_0 from Figure 2 could be implemented as p_0^t in the following manner:

$$\begin{array}{c} p_0^t = ((x_1 < 1),\ goto(table), [x_1 := 0]); (true \rightarrow (x_1 < 5)); \\ ((x_1 \geq 5),\ lift(table), [x_1 := 0]); p_0^t, \end{array}$$

which tells the agent to repeatedly go to the table within 1 time unit and lift it after 5 time units. We make the short remark that the timed choreography from Figure 1 gives enough flexibility to allow, for instance, the second agent to go to the table in 3 time units. However, it must lift the table within 6 time units, synchronously with the first agent.

We note that if previously actions failed when certain beliefs did not hold, it is now the case that actions fail also when certain clock constraints are not satisfied. Consider that the programmer forgets the delay action $(true \rightarrow (x_1 < 2))$ in p_0^t. It is then the case that time does not pass and x_1 remains reset to 0. This implies that lifting the table can never be achieved, which was not the intention. Such situations are handled by means of the general semantics of the repair rules. There are two possibilities: either to execute an action with a time constraint that holds, or to make time elapse. The latter is achieved by triggering a repair rule like $true \leftarrow \delta$, where for example δ is a delay action $true \rightarrow true$ which allows an indefinite amount of time to pass.

The runs of a BUpL multi-agent system consist of the executions of individual agents "guided" by a timed choreography. Instead of the usual definition of trace semantics we use *maximal traces* where we explicitly encode un/successful terminations depending on whether the choreography reaches a final state. In this way we are able to reason about deadlock situations. It is also the case that we redefine refinement with respect to *intention traces* which record information not only about actions being executed but also about *intentions*, i.e., actions that are enabled to be executed. This allows us to reason about multi-agent system refinement in the presence of a timed choreography, meaning that we can be sure that the multi-agent system implementation does not deadlock when running with respect to a timed choreography if this is the case for the specification.

3. REFERENCES

[1] R. Alur. Timed automata. In N. Halbwachs and D. Peled, editors, *CAV*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer, 1999.

[2] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[3] L. Astefanoaei and F. S. de Boer. Model-checking agent refinement. In L. Padgham, D. C. Parkes, J. Müller, and S. Parsons, editors, *AAMAS (2)*, pages 705–712. IFAAMAS, 2008.

[4] G. Boella, J. Broersen, and L. van der Torre. Reasoning about constitutive norms, counts-as conditionals, institutions, deadlines and violations. In T. D. Bui, T. V. Ho, and Q.-T. Ha, editors, *PRIMA*, volume 5357 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2008.

[5] M. Dastani, D. Grossi, J.-J. C. Meyer, and N. Tinnemeier. Normative multi-agent programs and their logics. In *KRAMAS'08: Proceedings of the Workshop on Knowledge Representation for Agents and Multi-Agent Systems*, 2008.