Coordinating the Safe Execution of Tasks in a Constrained Multi-Agent System

Anna Ciampolini, Paola Mello, Paolo Torroni
DEIS, Università di Bologna
V.le Risorgimento 2, 40136 Bologna, Italy
{aciampolini, pmello, ptorroni}@deis.unibo.it

Evelina Lamma
Dipartimento di Ingegneria
Università di Ferrara
Via Saragat 1, 44100 Ferrara, Italy
elamma@ing.unife.it

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Design, Languages

Keywords

Multi-Agent Systems, Formalisms and logics, Logic programming, Service composition, Task Execution

1. INTRODUCTION

The metaphor of software components as agent societies is appealing for many reasons, that made it very popular, but there are some obstacles to its applications to problems such as software engineering. According to [2], 'there are two major drawbacks associated with the very essence of an agent-based approach: (1) the patterns and the outcomes of the interaction are inherently unpredictable; and (2) predicting the behaviour of the overall system based on its constituent components is extremely difficult (sometimes impossible) because of the strong possibility of emerging behaviour.' This could be a drawback not only for software engineering. In fact, the need for making agents 'predictable', and – for most applications – as deterministic as possible, is indeed in contrast with the concept itself of autonomy. Nonetheless, it is reasonable to believe that societies (of agents) can only exist as long as the individuals' autonomy does not represent a threat for the other individuals, and for the society in general. Therefore, the important counterpart of autonomy is represented by private constraints and public laws, that can make the agents, if not predictable, at least not colliding with each other needs and constraints. We call 'safe' the execution of tasks in an agent system that does not violate any of the constraints in the system itself.

Much work has been done towards safety ensuring in agent systems, especially for inherently open systems such as mobile agent systems, where mobile agents could move to remote machines. For instance, a typical problem in this area is, from the hosting system viewpoint, the problem of enforcing some authorization policy, and make sure that hosted external agents do not consume more resources than are allowed. From the mobile agent viewpoint, the problem is instead to ensure that it can have the right resources that it needs in order to carry out a certain task. In this paper, we tackle the problem of ensuring that the execution of tasks in a constrained multi-agents setting is consistent with respect to its constraints. We propose a formalism, whose syntax and operational semantics is detailed in [1], that we use to express the way agents can coordinate the requests of services, and to verify that they do not collide with each other's conditions. We propose some operators, that could be mapped, in a concrete implementation, into a library of the language(s) used to encode the agents in the system. It is possible to prove [1] that, if the abstract machine underlying the agent code implements such operational semantics, all requests involving integrity constraints are allowed only if no constraint in the system is violated. In doing that, each agent performs a consistency check only within its own (private) set of constraints. That is, we ensure the consistency of the constraints in the whole system, without each agent needing to disclose its constraints to other agents.

2. FORMALISM

An agent is a triple $< P, S, IC >$, representing a program P written in a language that allows the definition of *functions*, a set S of such functions called *services*, and a set IC of *integrity constraints*. The services $s \in S$ are annotated with pairs $< s, \delta >$, that represent the agent's denotation, i.e., the set $Den(A) \stackrel{def}{=} \{< s, \delta >: s$ is locally provided by A under the conditions δ and $\delta \cup IC_{\{A\}} \nvdash \bot\}$. We assume that $Den(A)$ is sound and complete with respect to the notion of local computation. We do not make assumptions on the syntax of the agent programs, services, and ICs, although we assume that the agents are provided with a mechanism, that we call 'local consistency check', that is able to determine if the constraints are violated.

AAMAS'02, July 15-19, 2002, Bologna, Italy.

Our formalism allows to express collaborative / competitive service requests; the operators that we use are the following: (*i*) $>$ is the *communication* operator; (*ii*) & is the *collaborative* coordination operator; (*iii*) ; is the *competitive* coordination operator; (*iv*) $\downarrow$ is the *local execution* operator. Such operators are part of *expressions*, enclosed in the agent programs, which are evaluated when such services are needed by the agents, in what we call a successful (or unsuccessful) *top-down derivation*. A *successful* top-down derivation for an expression F that describes the request for a service made by an agent A, and returns a set of conditions δ and a bunch B of agents dynamically involved in the service, can be traced in terms of a (finite) tree such that: (*i*) the root node is labeled by $A \vdash_{\delta}^{B} F$; (*ii*) the internal nodes are derived by using, backwards, the inference rules defined in [1]; (*iii*) all the leaves are labeled by the empty formula, or represent a successful local computation. For instance, a formula including a collaborative operator $A \vdash_{\delta}^{B} f \ \& \ F$ develops into three branches (sub-trees), one for proving *f*, another one for *F* and a last one for the consistency check. Due to lack of space, we will only provide here some examples. We label the agents $\mathbf{A_0} \ \ldots \ \mathbf{A_n}$. Let the following expression be enclosed in an agent program, say A_0:

$$\texttt{A0} \ > \ \downarrow \texttt{s1} \ ; \ \texttt{A1} \ > \ \downarrow \texttt{s2}$$

It means that A_0 must either perform a local service, s_1, *or* ask agent A_1 for service s_2. Should both service be available, possibly under different conditions, the system will select non-deterministically only one of them. Let us consider, now, the following expression, also embodied in agent A_0's program, representing a collaborative request composed of two different sub-requests, whose conditions must be coherent with one another:

$$\texttt{A1} \ > \ \downarrow \texttt{s3} \ \& \ \texttt{A2} \ > \ \downarrow \texttt{s4}$$

Agent A_0 asks agent A_1 for the service s_3 and A_2 to for the service s_4; after both A_1 and A_2 reply to A_0 by giving each a set of conditions for the requested service, the result is obtained by merging such sets of conditions in a unique consistent set, with respect to the bunch of agents (A_0, A_1, and A_2) dynamically considered along the computation. Such set could be bigger than the union of the parts, due to additional constraints that are fired in the cross checking phase. The key concept, in the process of ensuring safe execution, is that of consistency derivation. We also define a concept of 'local consistency', and adopt the following notation: $A \overset{l-cons}{\rightsquigarrow}_{\delta} \Delta$, where Δ is a set of conditions. By 'local consistency' we mean that Δ is consistent with agent A's integrity constraints, IC, returning a (possibly enlarged) set of conditions δ.

3. RESULTS

The formalism we propose comes together with some results, that hold provided that the following two properties also hold for the consistency derivation.[1]

Property 1 Given an agent A_i and a set of conditions δ,

$$A_i \overset{l-cons}{\rightsquigarrow}_{\delta_i} \delta \Rightarrow \delta_i \cup IC_i \not\vdash \bot$$

where IC_i represents the integrity A_i's constraints.

[1]Although our system does not rely upon a logic system, there exist in literature some proof procedures that implement a local consistency derivation, such as [3], some of which are proved sound and complete.

This means that, if there exists a local consistency derivation for δ in A_i that returns a set of conditions δ_i, then δ_i is consistent with the integrity constraints of A_i itself.

Property 2 Given an agent A_i and a set of conditions δ,

$$\forall \delta : \delta \cup IC_i \not\vdash \bot \Rightarrow \exists \delta_i : A_i \overset{l-cons}{\rightsquigarrow}_{\delta_i} \delta \wedge \delta_i \supseteq \delta$$

where IC_i represents the integrity A_i's constraints.

This means that, if a set of conditions δ is consistent with the integrity constraints of an agent A_i, then there exists a local consistency computation for δ in A_i itself, which possibly adds some new conditions to δ, returning $\delta' \supseteq \delta$.

It turns out that, for any successful top-down derivation, under the assumption of soundness of the local consistency derivations (Prop. 1), the computed conditions satisfy all the integrity constraints of the bunch of agents dynamically involved in the service in question. This implements in a sense the introduced notion of *safe execution*.

Theorem 1 Let A be an agent and F an expression that possibly describes the request for a service. If there exists a *successful top-down derivation* for F in A, which returns a set of conditions δ and the bunch of agents B, then the computed set δ is consistent with the integrity constraints of the bunch B. Formally: $A \vdash_{\delta}^{B} F \Rightarrow \delta \cup IC_B \not\vdash \bot$.

It is also possible to prove a completeness theorem for a subset of the possible expressions occurring in a top-level request. The completeness theorem ensures that, if a δ can be found, satisfying the integrity constraints of a bunch of agents, and it belongs to the meaning of some agent, as defined below, then there exists an expression F and a successful top-down derivation for F in A leading to the bunch B of agents, and to conditions δ.

Theorem 2 Let B be a bunch of n agents, B, s a service and δ a set of conditions.
$\forall B = \{A_1, \ldots, A_n\}, \forall \ < s, \delta \ > \in Den(A_i), A_i \in B : \delta \cup IC_B \not\vdash \bot \Rightarrow \exists F, A \in B : A \vdash_{\hat{\delta}}^{B} F \wedge \hat{\delta} \supseteq \delta$ where F is an expression occurring in a top-level request.

In a similar way it is possible to give the proofs referring to other kinds of formula. We aware that this work tackles only part of the problems involved in multi-agent systems, but our approach has the characteristics that could allow its future integration in a more comprehensive multi-agent setting. In the future, we intend to provide a concrete implementation of the above described operators, and write a library that extends a virtual machine. We would like to make some experiments, and check if our idea represents a viable approach. We would also like to study the complexity implications of such mechanisms.

4. ACKNOWLEDGMENTS

This work was partially supported by the SOCS project, funded by the CEC, contract IST-2001-32530.

5. REFERENCES

[1] A. Ciampolini, E. Lamma, P. Mello, and P.Torroni. Coordinating the safe execution of tasks in a constrained multi-agent system. Tech. Rep. DEIS-LIA-01-009, 2001.

[2] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.

[3] A. C. Kakas and P. Mancarella. Generalized stable models: a semantics for abduction. In *Proc. 9th ECAI*, 1990.