Specification, simulation and verification of negotiation protocols in a unified agent-based framework

Extended Abstract

Daniela Briola, Maurizio Martelli, Viviana Mascardi
DISI - Dipartimento di Informatica e Scienze dell'Informazione, Università degli Studi di Genova
{daniela.briola, maurizio.martelli, viviana.mascardi}@unige.it

Abstract: In this paper, we describe a framework for specifying, simulating and verifying negotiation protocols following an agent-based approach. Most of the components of this framework have already been implemented and tested, whereas verification facilities using temporal logic, which are the most innovative aspect of this proposal, are being designed at the time of writing.

1. Introduction

Multi-Agent Systems (MASs) originate from contributions coming from different research fields: Artificial Intelligence, object-oriented languages and expert systems were the main ones. Starting from 1990, "Agent Oriented Programming" [1] was recognized as a discipline on its own, and nowadays it is a successful and lively multidisciplinary research field, spanning different domains.
MASs are widely used to model and develop complex systems where autonomous entities need to interact in order to solve a problem or to achieve a complex goal. An agent is an hardware or software system situated in an environment that it perceives and modifies, autonomous, capable of flexible actions, reactive, proactive and social [2]. Besides being employed as a *metaphor to describe and model complex systems*, the intelligent agents technology also provides *tools for implementing such systems*. More recently, agents and MASs started to be exploited for *simulating the emerging behavior of a system* made up of a large amount of actors: we refer to this third application of agents and MASs as "Agent-based modeling and simulation of social/biological domains" [3].
Our research is focused on modeling and simulating the behavior of complex systems where a limited set of agents must cooperate (or compete) to share a limited set of resources under topological and spatial constraints. This type of problems has been defined "Multiagent Resource Allocation (MARA) problems" [4]. We have designed an interaction protocol that either finds a sub-optimal solution to the MARA problem described in the sequel in a finite amount of time (in the order of minutes), or informs the user that it cannot find a solution.
Although in this paper we concentrated on the specific MARA problem that we were asked to solve by Ansaldo STS, the methodology we followed for solving it and the tools we used (or that we mean to use in the future) could be as well exploited for different problems and scenarios. For this reason, we think of the JADE platform extended with visualization capabilities thanks to NetLogo and with verification ones thanks to Concurrent MetateM as a *unified agent-based framework for the specification, simulation and verification of negotiation protocols.*

2. Description of our MARA problem

Our MARA problem derives from a case study provided to us by Ansaldo STS, a technology company operating in the global Railway & Mass Transit Transportation Systems business with the provision of traffic management, planning, train control and signalling systems and services.
Although the case study refers to a very concrete application of Ansaldo STS, we had to generalize it in the following way because of a signed Non Disclosure Agreement.
The generalized problem consists of:

- A set of indivisible resources that must be assigned to different entities in different time slots (each resource can be used by only one entity in each time slot).

- A set of entities with different priorities, each needing to use some of the available resources for one or more time slots; entities have preferences over the set of resources they can obtain.
- A direct non planar graph of dependencies among resources: if an entity can use resource R only if it used a resource from {R1 , R2 , ..., Rn} in the previous time slot, then we represent these dependencies as arcs R1 → R, R2 → R, ..., Rn → R in the graph.
- A set of resources named "start points" that can be assigned to entities without requiring the prior usage of other resources (no arc enters in the corresponding node).
- A set of resources named "end points" that, once assigned to one entity, allow the entity to complete its job (no arc exits from the corresponding node).
- A set of couples of conflicting arcs in the graph of dependencies: an entity releasing R1 for accessing R2, where the usage of R2 depends on the previous usage of R1, might conflict with an entity releasing R3 for accessing R4. The two entities might indeed need to use the same transportation means for accessing R2 from R1 and R4 from R3 respectively, and the transportation means might be non shareable as well.
- A static allocation plan that assigns resources to entities for pre-defined time slots, in such a way that no conflicts arise.

Since we are going to simulate what happens the real world, where entities happen to use resources for longer than planned and where resources can break up, a dynamic reallocation of resources over time is often required. Thus, the interaction protocol we designed to allow agents to compute a re-allocation when the static allocation plan cannot be satisfied, has the following features:

- the re-allocation is feasible, namely free of conflicts; in our scenario, conflicts may arise both because two or more entities would want to access the same resource in the same time slot, and because two or more entities would want to use conflicting arcs in the same time slot;
- the re-allocation task, whose output is either a feasible reallocation or a message stating that no feasible re-allocations exist, is completed within a pre-defined amount of time;
- each entity minimizes the changes between its new plan and its static allocation plan: the start and end point must always remain those stated in the static allocation plan, but the nodes in between may change, as well as the time slots during which resources are used;
- each entity minimizes the delay in which it reaches the end point with respect to its static allocation plan;
- the number of entities and resources involved in the re-allocation process is kept to the minimum;
- in case no feasible re-allocation exists, the algorithm stops.

3. Design

In order to face the dynamic resource allocation problem we designed and implemented a MAS characterized by three types of agents each with its own capabilities and view of the system: Resource Agents (RAs), User Agents (UAs), Interface Agents (IAs). The graph becomes the Environment where agents live. There is no central control of the state of the graph, which is indeed spread all over the RAs.

Resource Agent. Each node in the graph is managed by one RA. RAs do not take decisions about which UA will obtain the control of the node but keep track of the node's state (free/occupied). RAs also manage the allocation of arcs entering the node.

UAs interact with RAs for knowing whether the node is free or occupied in a given time slot. RAs answer the question and, in case the node is not free, tell which UA occupies the node for the given time slot, its priority, and when the node will become free again.

RAs also manage the allocation of the arcs incoming into the node they control. The RA controlling node N has the list of all its neighbors, namely those RAs controlling nodes N^{from} such that an arc $N^{from} \rightarrow N$ exists. For each arc $N^{from} \rightarrow N$, the RA also possesses the list of arcs A1 , ..., Ak that conflict with $N^{from} \rightarrow N$.
When the RA receives a reservation request for node N and chooses the free arc $N^{from} \rightarrow N$ to reach it, it updates its reservation table by marking the arc as occupied, answers the request, and informs all the RAs that may be interested by this reservation, namely those controlling arcs A1 , ..., Ak, about the new state of the arc.
These RAs need to know that arcs conflicting with $N^{from} \rightarrow N$ can not be used for the specified time slot. In this way, the neighbours of RA have up-to-date information about the state of the arcs that might cause conflicts with their own arcs, and will be able to provide conflict-free answers to successive reservation requests.
User Agent. UAs represent entities that want to traverse the graph. Each UA has an original plan stated in the static allocation plan and consisting of the list of nodes to traverse together with arrival and departure time for each of them. As soon as an UA enters the graph, no matter if it is on time or late, it always tries to get a reservation for the nodes in its original path. UAs do not try to reserve a specific arc to reach a node: they ask RAs to reserve the most suitable arc for them.
UAs do not know the topology of the graph; they may interact with the Path Agent (introduced later) to obtain information on the paths that connect the start point where they enter the graph with the end point they must reach to exit the graph. UAs communicate with RAs to reserve resources.
Every UA has a priority that it uses to reserve a resource or even to steal a resource to another UA. Since each UA has the unique goal of getting out of the graph, it continues to look for a path in it until it obtains the reservation for all the nodes in the path. If one, or more, nodes are already reserved by one other UA, the UA can steal the resource if it has a higher priority. In this situation the UA that lost the resource must search for another path in the graph.
If an UA looses the reservation of one of these nodes, for example because a UA with higher priority stole the node to it, it will start the search again until it will succeed in reserving all the nodes in one path.
Interface Agent. IAs act as an interface between the MAS and the external environment. There are different types of IAs, but the most interesting for the protocol is The **Path Agent** (PA), that exploits its knowledge about the graph topology and geometry. Given two nodes, it returns a list of selected paths connecting them ordered from the best one to the worst one. The strategy for selecting and ordering the paths depends on the application. In Ansaldo-STS's application, it depends on the number of nodes in the path and on geometrical constraints.

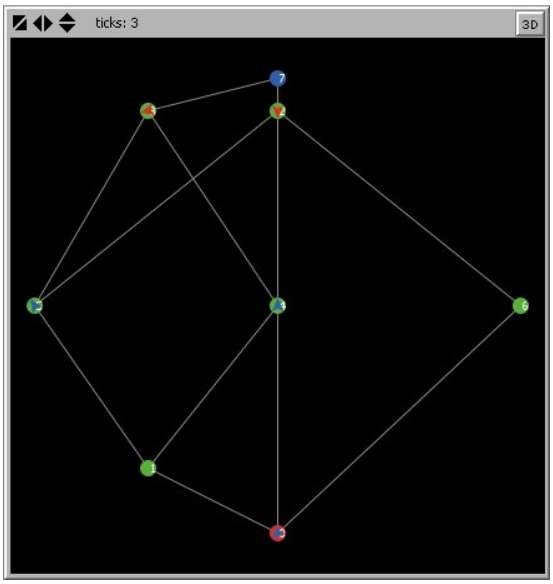

Figure 1: NetLogo GUI showing UAs (triangles) that must reach the end node with the same color.

4. Implementation in JADE and NetLogo

The system that implements the architecture above, as well as the complex interaction protocol for resource re-allocation that we just sketched and that we could not describe in detail for space constraints, have been developed using JADE [5], a platform for building MASs compliant to the FIPA standard [6] that offers a ready-to-use infrastructure for message exchange. Since JADE offers no built-in graphical interface, we added an "off-line" interface for showing the way agents move in the graph, after the JADE simulation stops. During their execution in JADE, all agents create a log file. We realized a program in the NetLogo tool [7] for Agent-Based Modeling and Simulation that

reads the configuration of the graph and the agent's log files and shows how UAs moved on the graph during the time (Figure 1). NetLogo offers a graphical interface "for free", that is, the NetLogo programming language includes commands to graphically represent agents, their environment, and the way they move in it. Integrating NetLogo GUI in JADE in such a way that visualization of UAs moves was on-line instead of off-line, raised many technical problems that, although easy to solve from a theoretical point of view, should have required too much time and effort from us. Hence, we opted for the off-line visualization that gave satisfactory results without requiring too much effort and time.

5. Verification using temporal logic

Verification of MASs using temporal logic raised a lot of attention starting from the early nineties. In order to make our systems an *agent-based framework*, we plan to integrate verification capabilities offered by Concurrent MetateM [8] into it.

Concurrent MetateM is a multi-agent language in which each agent is programmed using a set of (augmented) temporal logic specifications of the behaviour it should exhibit. These specifications are directly executed in order to generate the behaviour of the agent. As a result, there is no risk of invalidating the logic as with systems where logical specification must first be translated to a lower-level implementation. Two prototypical implementations of the Concurrent MetateM interpreter exist. The root of the MetateM concept is Gabbay's separation theorem; any arbitrary temporal logic formula can be rewritten in a logically equivalent *past* => *future* form. Execution proceeds by a process of continually matching rules against a history, and firing those rules when antecedents are satisfied. In order to verify MASs using temporal logic, agents should be specified in the Concurrent MetateM language instead of in natural language and then directly programmed in JADE, as it happens now. We are exploring the use of the Prometheus Design Tool for MASs (http://www.cs.rmit.edu.au/agents/pdt/) as a suitable means for specifying agents and their interactions in an high-level, declarative graphical language, and then translating this specification into both a Concurrent MetateM one for verification, and into JADE agents for simulation. We faced a similar translation problem in the recent past [9], and we intend to apply an approach similar to the one discussed there, which proved to be successful.

6. Conclusions

In this extended abstract, we described our experience in specifying, simulating and visualizing the behavior of agents in a MAS, engaged in an interaction protocol for solving a complex MARA problem. We also outlined the intended use of existing techniques for verifying MASs using temporal logic and how we would like to integrate them into our system, in order to make it a *unified agent-based framework for the specification, simulation and verification of negotiation protocols*.

Bibliography

1. Y. Shoham, "Agent-oriented programming", Artificial Intelligence 60(1)
2. N. Jennings, K. Sycara, M. Wooldridge, JAAMAS 1(1), 1998
3. C. M. Macal and M. J. North, Proc. of Winter Simulation Conference, 2007
4. Y. Chevaleyre, et al. Issues in multiagent resource allocation. Informatica, 30:3–31, 2006
5. F. L. Bellifemine, G. Caire, and D. Greenwood. Developing Multi-Agent Systems with JADE. Wiley, 2007
6. FIPA http://www.fipa.org/
7. NetLogo. http://ccl.northwestern.edu/netlogo/
8. M. Fisher. A Survey of Concurrent METATEM - the Language and its Applications. In Proc. of the First international Conference on Temporal Logic, 480-505, 1994.
9. G. Casella, V. Mascardi: West2East: exploiting WEb Service Technologies to Engineer Agent-based SofTware. IJAOSE 1(3/4): 396-434, 2007.