Model checking applied to mobile agent system

Cyril Dumont, Fabrice Mourlin
LACL, Paris East University, 94100 Creteil, France
dumont_cyril@yahoo.fr, fabrice.mourlin@wanadoo.fr

***Abstract*—The subject of this contribution is about use of formal specification in order to prove business properties of a mobile agent system. First, we can be described this kind of system by the use of process algebra, in particular the π-calculus language. Then, from such formal specification, we can generate semi automatically automata using an XML transformation. Finally, we compute acceptance graph to validate a formal property. This validation is assisted by the use of a tool such Uppaal from Aalborg university.**

***Index Terms*—mobile agent system, process algebra, temporal logic, model checking.**

I. INTRODUCTION

Mobile agent systems provide a new approach of software based on adaptability of the system against its runtime context. Also, this means mobile agent is a reactive component like a stimulus recorder device. For example, a mobile agent can interact with resource to extract data. If the resource is missing, it can adapt its behaviour and move on other computer to perform its task.

The relative informality and high level of abstraction of current practice in describing mobile agent system might at first glance suggest that mobile agent descriptions have little substantive value for software engineers. First, over time engineers have evolved a collection of idioms, patterns, and styles of agent system organization that serve as a shared, semantically rich vocabulary between engineers.

Formal models are needed for careful description and reasoning about agents, just as for other kinds of distributed systems. Important issues that arise in modeling agent computation are: agent communication, dynamic creation and destruction of agents, mobility, and naming. How to choose a formal approach ? There are multi factors and is based not only on application domain but also on past experience. And how can we check the specification from this formal approach ? Two questions we will try to answer in this paper.

II. FORMAL SPECIFICATION

Process algebra is widely used for studying concurrent processes (CCS: Milner's Calculus of Communicating Systems [1], ACP: Bergstra & Klop's Algebra of Communicating Processes [2]). But the expression of mobile agent needs use of languages which allow higher order term. Actually a agent is not just a name or a channel but a data structure. More recently, π-calculus language provides such kind of term and mobile agent are expressed as simple as a function call [3]. Several semantics are defined for this formal language and tools are built for creating new specification. Also, this language with extensions is selected for specifying our mobile agent system like the the Higher Order π-calculus (HOπ). Extensions are about polyadic communications and higher order term construction. Indeed in the HOπ-calculus not only names, but also agents of arbitrarily high order, can be transmitted.

This core language is rich enough to describe distributed protocol such as SLP (Service Location Protocol) [4] or detection intrusion system such as AAFID [5]. We have now experience about process algebra and more precisely with this formal language. Also, we built case studies and our formal specifications were used to drive project construction. This means how mobile agent can move from one computer to another one. This means also the structure of exchanged messages over the network.

Some programming logics have been proposed to express properties of π-calculus agents. We can cite the μ-calculus that allows to express a given property is verified by a given process (in π-calculus). Computer-assisted proof adapted to π-calculus are very few. However the *Mobility Workbench* allows to define agent and to verify some properties. Unfortunately some work as [6] show that, despite a important expressive power,the π-calculus and in particular its logics adapted have a lack of suitable model-checkers.

III. TEMPORAL PROPERTIES OF MOBILE AGENT SYSTEM

When mobile agent application is deployed over a network, it can be useful to check traditional assertions. This wish needs a specific approach where initial formal specifications are a basis.

Also we have examined the use of assertions in testing new and improved programming. The kind of properties can be about event occurrence such that specific communication between two agents on a given location. Of course, specifications define formal type for agents, messages, etc ...But, type checking does not cover all features. Temporal logic is a formalism used to describe how a program state will change with time.

In addition to type checking, assertions provide a great way to determine that various properties are maintained in a mobile agent application. Our interest concerns three categories of common assertion properties. Traditionally, assertion properties fall into one of these three categories. First, pre-conditions assert that a property holds before execution of a code block. Then, post-conditions assert that a property hold after execution of a code block and finally invariants assert that a property holds before and after the execution of a code block.

As helpful as assertions of these typical forms can be, they don't quite have the range for all the properties we'd like to be able to hold in a mobile agent system. This is just a short list of the types of system properties that can be expressed in a traditional assertion language (properties that we would like verify in agent behaviour when we modeling a system of mobile agents)

- To ensure that any location (i.e. a computer) is visited only once
- To assert that resources are never accessed by unauthorized agents
- To assert that each agent is given a chance to run
- To assert that the system will never get itself into deadlock (i.e. two or more agents are waiting on each other)

Following are two very useful types of properties that a specifier likes to make that are simply not possible to express with conventional assertions:

- Safety assertions state that certain undesirable states of the agent system will never be reached under any circumstances.
- Liveness assertions state that certain events are guaranteed to occur eventually, for instance, that a given agent will eventually wake up instead of sleeping forever.

Temporal logic can help make these assertions. This is a type of modal logic [7] that is used for reasoning about changing properties with time. Several modal operators are usually available in temporal logic: always, sometime, until, next.

Here's an example assertion for two agents that asserts they never deadlock. (Note that the boolean method *isWaiting* is used to check if one agent is waiting a task performed by the other.)

```
always {agent1.isWaiting(agent2)}
     implies {! agent2.isWaiting(agent1)}
```

The translation of this formulae is : in all cases during the execution of the mobile agent system , if *agent1* is waiting a task performed by *agent2* then *agent2* can't be waiting a task performed by *agent1*.

IV. FROM π-CALCULUS SPECIFICATION TO TEMPORAL LOGIC

To prove that the previous formulae is verified, it is essential to transform our type definitions (written in π-calculus)into automata definition. Also, we defined operators to build timed automata from an agent definition with a structure close to term definition. Because concrete specifications are quite large, these operators are developed as data transformers.

In our case of mobile agent system, our ability to statically check such assertions is paltry, but quality tools exist for checking that these assertions hold during particular runs of the program. We can use UPPAAL [8], where the two inputs to the model checking problem are the system model and the properties that such a system must satisfy.

We can cite the work done by the creators of the *HD Automata Laboratory* (HAL) described in [9]. The goal of the HAL is to verify properties of mobile systems specified on π-calculus. They pass by an intermediate step, *History Dependent automata* (HD-automata), to generate ordinary automata from π-calculus agents.

In our case, we will use the power of XML formalism. A choice of more evident that UPPAAL uses this formalism.So we use a XSL transformation (XSLT for eXtensible Stylesheet Transformation) that is an XML operation which transforms an input XML graph into another one. The input XML graph is one of agent definitions. The output XML graph is a skeleton of timed automata. The translation is not complete because of clock definitions and token type. These clocks should not be confused with hardware clocks (local to computer). Clocks should rather be considered as stop watches or chronometers. Time is continuous and all clocks advance at the same rate, though it is possible to test the value of a clock or reset it.

V. CONCLUSION AND FUTURE WORK

Our formal approach has allowed us to model our earlier work on an architecture based on mobile agent for numerical solving [10].From a code written in the Java language, we got to specify agents in HOπ-calculus. Translated into a XML graph, these agents have been transformed into automata (with XSLT). This modeling allowed us to verify if such a computation, deployed on our platform, will always end whatever number of agents available for this calculation (using UPPAAL software).

Next step of our work is to manage code generation with the use of formal specification. We first computed a set of automata from pi calculus representations by using congruent operator. We want now to define new operator to get a skeleton of mobile code. Finally, this next step should keep initial assertions into programming definition.

REFERENCES

[1] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.

[2] J. Bergstra and M. Loots, "Program algebra for component code," *Formal aspects of computing*, vol. 12, pp. 1–17, 2000.

[3] R. Milner, *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.

[4] J. Kempf, R. S. Pierre, and P. S. Pierre, *Service Location Protocol for Enterprise Networks: Implementing and Deploying a Dynamic Service Finder*. John Wiley & Sons, 1999.

[5] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni, "An architecture for intrusion detection using autonomous agents," COAST Laboratory, Purdue University, Tech. Rep., 1998.

[6] N. Bernard and Y. Dumond, "Spécification en pi-calcul de l'étude de cas relative au contrôle d'accès," in *Proceedings of AFADL'07*, 2000.

[7] E. A. Emerson and J. Y. Halpern, "sometimes and not never revisited: On branching versus linear time temporal logic," *Journal of the Association for Computing Machinery*, vol. 33, pp. 151–178, 1986.

[8] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *Int. Journal on Software Tools for Technology Transfer*, vol. 1, pp. 134–152, 1997.

[9] G. Ferrari, S. Gnesi, U. Montanari, and M.Pistore, "A model checking verification environement for mobile processes," *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 4, pp. 440–473, 2003.

[10] C. Dumont and F. Mourlin, "Space based architecture for numerical solving," in *CIMCA 2008: Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation*. Vienna, Austria: IEEE Computer Society, December 2008, pp. 309–314.