Application of ASM to the Specification of Mobile Systems

Marco Túlio de Oliveira Valente Roberto da Silva Bigonha
Marcelo de Almeida Maia

Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
30161-970 - Belo Horizonte - MG - Brasil
{mtov,bigonha,marcmaia}@dcc.ufmg.br

Abstract

In this paper we present a formal method for the specification of mobile systems using Abstract State Machines (ASM) [3]. The method is based on the Ambient Calculus [2], a process calculus developed to express mobility. In the work we show how the fundamental abstractions of the Ambient Calculus can be expressed in ASM without difficulty. In order to exhibit the feasibility of the proposed method, we also show as a case study an ASM specification of a mobile system for electronic commerce.

Keywords: mobile systems, formal methods, ASM

1 Introduction

Recently mobile systems are emerging as an alternative to increase the computational use of the Web. In the systems designed following this model, a computation can move from one network host to another, where for example the resources needed to perform its task are locally available. It is argued that this kind of system reduces network load and latency and makes it easier to design more robust and fault tolerant systems [4]. Numerous packages are currently available to deploy mobile systems, most of them based on Java. Among those packages, we can mention IBM Aglets and General Magic Odyssey (formerly known as Telescript).

As this computational model becomes a reality, it arises the need to develop formal methods for the specification of mobile systems. Through these formalisms, it will be possible to correctly understand the behavior of those systems, to prove properties about them and also to design new environments and programming languages to support their development. Recently, some formal methods are being proposed with this intention. Among them, the Ambient Calculus [1, 2] is the most prominent one, probably because it has been originally designed having in mind the specification of mobile computation.

In this work, we show that the fundamental abstractions for mobile computation presented in the Ambient Calculus can be mapped without difficulty to Abstract State Machines (ASM) [3, 7]. ASM is a formalism that has been successfully used to specify a variety of systems, including programming languages, computer architectures and distributed and real time systems. We hope that with the method proposed in this paper ASM can also start to be used in the specification of mobile systems.

This paper is organized as follows. In section 2, the main ideas proposed by the Ambient Calculus are described. Section 3 presents the Abstract State Machines and section 4 describes how the main abstractions of the Ambient Calculus can be mapped in ASM. In section 5, we show as a case study the specification in ASM of a mobile system for electronic commerce. We also prove in this section two propositions about this system. Finally, section 6 concludes the paper and relates future work.

2 Ambient Calculus

Being the Web in large scale a global distributed system, there are some proposals to express mobility in the Web using the already established formalisms to model distributed and concurrent systems. Among these formalisms, one of the most used is the π-calculus [6], which is based in the notion of processes communicating over channels, with the ability to create new channels and to exchange channels over other channels. However, such facilities are not enough to capture the notion of mobility as it is used in mobile systems. In those systems mobility is related with a change in the execution environment of a process and not in its number of channels. In this sense, it seems more correct to say that in π-calculus we have channel mobility and not process mobility.

Inspired in the π-calculus, the Ambient Calculus [1, 2] has been proposed as process calculus that focuses primarily on process mobility rather than process communication. An ambient is a bounded place that has a name and that may contain processes and subambients. An ambient can already move inside or outside another ambient. This property is regulated by capabilities that the processes must possess. Being a bound place with established borders, it is easy to determine what is moved together with an ambient. Then in the Ambient Calculus, mobility is associated with the notion of crossing barriers that delimit ambients, which can be hierarchically organized, forming a tree structure.

3 Abstract State Machines

Abstract State Machines [3, 7] are a computational model where any sequential algorithm can be specified in its natural abstraction level. In ASM, a state S is an algebra defined by a vocabulary Υ of function and relation names, a set X called the superuniverse and a interpretation function Val of vocabulary names into functions $X^* \to X$. Transition rules are used to modify the interpretation of names from one state to another. The execution of an ASM program is a sequence of transition rules fires that changes the machine state.

In ASM there are three basic transition rules: an update rule, a conditional rule and a block construction rule.

An update rule has the form $f(t_1, \ldots, t_j) := t_0$, where j is the arity of f and $t_0, \ldots, t_j$ are terms. The fire of this rule in a state S, where the terms $t_0, \ldots, t_j$ are evaluated to $a_0, \ldots, a_j$ respectively, gives a new state S' where the function interpreted by the name f has value a_0 in the point $a_1, \ldots, a_j$.

A conditional rule has the following form: **if** φ **then** R_1 **else** R_2 **endif**, where φ is a boolean term and R_1 and R_2 are rules. The semantics of this rule is trivial: if the term φ evaluates to *true*, then the next state is the result of firing rule R_1; otherwise, it is the result of firing R_2.

Finally, a block construction rule has the form $R_1, \ldots, R_n$ and the following semantics: the next state is the result of firing all the rules R_i *in parallel*.

An ASM specification defines a initial state S_0 and a transition rule R. The execution of a specification is a sequence of states $\langle S_n : n \geq 0 \rangle$, where each S_i is obtained firing the rule R in S_{i-1}.

4 Specification of Mobile Systems in ASM

The main abstraction for mobile computation proposed in Cardelli's work was the notion of ambient, defined as a "bounded place where computation happens"[1]. In ASM we also have the notion of ambient (called environment in the method definition), but with the aim of making available external functions used by the machine in its computation[3]. Among other applications, external functions are used to model input/output, to express non-determinism and to provide information hiding.

In this work, we propose that ASM environments, from now named ambients, are used not only to provide external functions but also as a reference location for the computation executed by the machine. Then we propose that mobility in ASM can be characterized as the capability to change the state of a machine (i.e., its vocabulary Υ, its superuniverse X and its interpretation function $Val_S : \Upsilon \rightarrow X^* \rightarrow X$) from an ambient to another.

We still propose that besides external functions an ambient also has the following characteristics:

- Each ambient has a name, that is used to control access to the ambient.
- Each ambient has a set of ASMs.
- Each ambient has a set of subambients, i.e., ambients are hierachically structured.
- Each ambient provides an atomic operation called *move* to the machines in its boundary.

Suppose that the machine M is in an ambient m and its current state is S_i. Then the fire of a transition rule including the operation *move* n makes the next machine state, S_{i+1}, to be located in the ambient named n. If this ambient is not available, the execution of M becomes locked in S_i until the transition to S_{i+1} can occur. Therefore this external operation adds the notion of state mobility to ASM, where the notion of state is the same as in the original definition of the method.

Because that with mobility an ASM computation can roam over ambients, we define that the binding between the call of an external functional and its availability in an ambient is totally dynamic.

To make it possible for an ASM to know its current execution environment, we also introduce a zero argument function called *here*, which returns the name of the current ambient where the machine is executing.

The introduction of ambients in ASM is described in a formal approach in [5].

5 Case Study

We show below the specification of a mobile system for electronic commerce. This system searchs the price of a certain book in a collection of Internet bookstores. We suppose that the system starts its execution in an ambient named `home` and then roams over a set of bookstores, each of then represented by an ambient. In each bookstore the system (an ASM computation) locally searchs the price of the book and, if found, stores the price in its state. After visiting the last bookstore, the system returns to `home` ambient, where it locally determines which bookstore has the lowest price. The specification below is done using the language described in [8].

```
ambient home;

machina book_search_agent;

external

number_books: Integer;              // Total number of available books in the bookstore
book_list (Integer): String;        // List with the names of the available books
price_list (Integer): String;       // List with the price of the available books
book_requested: String;             // Name of the book requested by the user

vocabulary

Ambient;                            // Universe of ambient names
book_name: String;                  // Name of the book to search for
bookstore: List of Ambient;         // Bookstores to visit
price: Ambient → Real;              // Price of the book in each bookstore
status: enum { initial, travelling, searching, found, not_found, final };
index: Integer;

init

book_name=:= book_requested;
bookstore:= [ Amazon, Bookpool, BarnesAndNobles ];
status:= initial;

rule

if (status = initial) then                                              // rule 1
    status:= travelling, bookstore:= tail (bookstore), move head (bookstore)
elsif (status = travelling) then                                        // rule 2
    status:= searching, index:= 1
elsif (status = searching) then                                         // rule 3
    if index > number_books then
```

```
            status:= not_found
        elsif book_list (index) = book_name then
            price (here):= price_list (index), status:= found
        endif,
        index:= index + 1
elsif (status = found) or (status = not_found) then                    // rule 4
        bookstore:= tail (bookstore),
        if head (bookstore) = [] then                                  // rule 4.1
            status:= final, move home
        else
            status:= travelling, move head (bookstore)
        endif
elsif (status = final)                                                  // rule 5
        // Transitions to search for the lowest price found in the "travel"
end;
```

We prove below two properties about this system.

Proposition 1 *The search executed by the system in a certain bookstore always finish.*

Proof: We need to prove that if $Val_{S_i}(status) = \texttt{searching}$, there is a $j > i$, such that $Val_{S_j}(status) = \texttt{final}$ or $Val_{S_j}(status) = \texttt{travelling}$.

Suppose that $Val_{S_i}(status) = \texttt{searching}$. In this case, the rule 3 will be fired. This rule can change the value of *status* by two ways: (i) when the book is found, the value of *status* changes to `found`; (ii) when the book is not found, the value of *status* changes to `not_found`. As in every transition a new book is inspected, we have that in some state S_k, case (i) or case (ii) will occur, and then $Val_{S_k}(status) = \texttt{found}$ or $Val_{S_k}(status) = \texttt{not\_found}$. So, in S_k the only rule that can be fired is the rule 4, which produces a state S_j where $Val_{S_j}(status) = \texttt{final}$ ou $Val_{S_j}(status) = \texttt{travelling}$. □

Proposition 2 *In the absence of locks, the search in all the bookstores finishes and the system returns to* `home` *ambient.*

Proof: We need to prove that if $Val_{S_0}(status) = \texttt{initial}$ and $Val_{S_0}(here) = \texttt{home}$, then there is a $j > 0$ such that $Val_{S_j}(status) = \texttt{final}$ and $Val_{S_j}(here) = \texttt{home}$.

In the initial state S_0, by direct inspection in the specification text, we have that $Val_{S_0}(status) = \texttt{initial}$ and $Val_{S_0}(here) = \texttt{home}$. Then the rules 1 and 2 are fired and we have a state S_2, where $Val_{S_2}(status) = \texttt{searching}$. By proposition 1, the system will reach a state S_{j_1}, where two cases can happen:

1. $Val_{S_{j_1}}(status) = \texttt{final}$: In this case, we have $Val_{S_{j_1}}(here) = \texttt{home}$, because the only rule that can be fired to reach this state is rule 4.1. So the proposition is verified.
2. $Val_{S_{j_1}}(status) = \texttt{travelling}$: In this case, the state S_{j_1} was reached by firing rule 4, which also removes an ambient from the bookstore list. Next, we fire rule 2, reaching a state S_{i_2}, where $Val_{S_{i_2}}(status) = \texttt{searching}$. Then, by proposition 1, we have case 1 above or this own case again.

By induction on the length of the bookstore list, we have that in some state S_j, the case 1 will be choose and the proposition will be verified. □

6 Conclusions

In this paper we show a formal method to the specification of mobile systems using Abstract State Machines and inspired in the main abstractions for mobile computation proposed in the Ambient Calculus.

Compared against the Ambient Calculus, the specification of mobile systems in ASM has the following benefits:

- ASM is a formal method easy to learn and to use, requiring only basic mathematical knowledge of the users.
- ASM has already been used to specify a variety of systems. We hope that with this paper they can also start to be used in the specification of mobile systems.
- ASM specifications can be directly executed, allowing the user not only to specify a system but also to simulate its behavior.

Formal specification of mobile systems is a novel research area and therefore offers many possiblities of further work, focusing problems not analysed in this paper. Among them we can include security, communication between mobile systems, exception handling and type systems to control mobility.

References

[1] Cardelli, L. *Abstractions for Mobile Computations.* Technical Report, Microsoft Research, 1999.

[2] Cardelli, L. and Gordon, A.D. *Mobile Ambients.* In *Foundation of Software Science and Computational Structures*, Maurice Nival (ed.), Lecture Notes in Computer Science 1378, Springer, 1998.

[3] Gurevich, Y. *Evolving Algebras 1993: Lipari Guide.* In *Specification and Validation Methods.* E. Börger (ed.), Oxford University Press, 1995.

[4] Lange, D.B. and Oshima, M. *Seven Good Reasons for Mobile Agents.* Communications of the ACM, 42(3), March 1999.

[5] Maia, M.A., Bigonha, R.S., Valente, M.T.O. *A proposal for the formalization of ambients in ASM (in portuguese).* Submitted to 2nd Brazilian Workshop on Formal Methods, October 1999.

[6] Milner, R.J., Parrow, J. and Walker, D. *A Calculus of Mobile Process.* Parts 1-2, Information and Computation, 100(1), 1992.

[7] Tirelo, F., Maia, M.A., Di Iorio, V. e Bigonha, R.S. *Abstract State Machines (in portuguese).* III Brazilian Symposium on Programming Languages, May 1999.

[8] Tirelo, F., Bigonha, R. S., Maia, M.A. and Di Iorio, V. *Machina: The ASM specification language (in portuguese)* Technical Report 08/99, Programming Language Laboratory, UFMG, June 1999.