Simulation of Mobile Applications[1]

Fernando Luís Dotti, Lucio Mauro Duarte, Bernardo Copstein
[fldotti, lduarte, copstein]@inf.pucrs.br
PUCRS – Faculdade de Informática – PPGCC
Av. Ipiranga, 6681 – Prédio 16 – ZIP 90619-900
Porto Alegre – RS – Brazil

Leila Ribeiro
[leila]@inf.ufrgs.br
Instituto de Informática – UFRGS
Av. Bento Gonçalves, 9500 – Bloco IV – ZIP 91509-900
Porto Alegre – RS – Brazil

Keywords: Mobile and Nomadic Computing; Mobile Applications; Distributed Systems.

Abstract

Mobile applications are composed by software components that may migrate through the nodes of a network. Migration is controlled by the software developer, being part of the applications functionality. There are various potential application fields for this new technology. However, developing mobile applications is a difficult task since they are not only distributed and concurrent, but also mobile. This increases the complexity of testing in real conditions, where one cannot be sure whether an error is a result from the mobile application or from the environment in which it runs (e.g. the Internet). Simulation is used in this work as a means for testing models of mobile applications, such that the developer may check if the model behaves as expected under known environment conditions that can be chosen. We use a formal specification language for mobile applications that is based on graph grammars. The entities of a model according to this formalism can be simulated and code can be generated to build a real application. This paper describes the specification formalism, the code generation approach, and the simulation environment in more detail. Parts of a specification of a mobile application that has been simulated and executed are given as examples.

1 INTRODUCTION

The fast and continuous growth in processing and communication capabilities led to massively distributed computational environments, e.g. the Internet. These environments are often called open environments, being characterized by: massive geographical distribution; high dynamics (appearance of new nodes and services); no global control; partial failures; lack of security; and high heterogeneity [2]. Building distributed applications for such environments is a complex task. Research efforts have been directed to manage this complexity through the development of new paradigms, theories and technologies for distributed applications. Within this context, code mobility [9] has received special attention due to its flexibility and potential use in various application fields [2].

In traditional distributed systems, interacting components are primarily static, i.e., a component remains on the same location during its whole life-cycle and interactions with other components take place by exchanging messages through the communications network [2]. Code mobility can be defined as the capability of dynamically changing the location of an executing component. Migration is not transparent to the distributed software developer, it is instead explicitly handled by him/her as part of the application's functionality. Under the various paradigms enabled by code mobility [2], the Mobile Agent one is very promising and also investigated because of the agents mobility autonomy: a mobile agent is able to stop its execution on a location, migrate through the network carrying its internal state, and resume its execution on another location. Currently there are standards, platforms and languages available for mobile code [19]. However, there are still problems to be addressed in order to build a sound support for mobile code. Most of these issues are related to what is called controllability of agent-based activities in [2], involving support to the generation of correct mobile code applications and support to the reliable execution of mobile applications.

Simulation models may be very useful for the development of mobile applications as a specification and testing means. These models may be used to check if the components of an application behave as expected, if they are independent from each other such that the replacement of a simulated component by a more sophisticated version becomes possible, and also to check the applications behavior under various environment conditions (e.g. failure simulation). The main advantage of developing a mobile application using a simulation model is the possibility to validate strategies as well as control algorithms. If the simulation model is described using formal specification there is also the possibility of performing verification techniques to guarantee that the required properties of the system are fulfilled.

In the context of project ForMOS we are following an approach for the development of correct mobile applications that relies very much on simulation, among other methods and tools. Our approach involves: a formal specification language with abstractions that allow the representation of places and mobile agents (and their inherent mobility across places) [7]; a straightforward mapping of models written according to this formalism to entities of a simulation environment; the event driven simulation environment, which supports the abstractions of places and mobile agents; and another mapping of models written using this formalism to entities of a real application, running on top of a mobility support platform [8]. Verification tools and methods are under investigation to be integrated to our approach.

In this paper we briefly present the specification formalism and the code generation facility for mobile applications specified following this formalism; and discuss in more detail about the importance of simulation for mobile applications, as well as present a simulation environment and some simulation cases with mobile applications.

This paper is organized as follows: Section 2 presents the specification formalism; Section 3 discusses the simulation environment; Section 4 presents the code generation approach, through which we generate mobile applications to run on top of

[1] This work is partially sponsored by ForMOS (CNPq, CNPq/ProTeM, FAPERGS) and Platus (FAPERGS) projects.

a commercial mobility support platform; finally, Section 5 brings us to the conclusions and future works.

2 FORMAL SPECIFICATION OF MOBILE APPLICATIONS

When considering mobile code applications, complex distribution aspects, like location and mobility, communication, security, and, in some cases, failures, have to be taken into consideration during system design. Therefore, it is necessary to provide the designer with abstract constructions to specify and reason about those aspects. Starting with the Pi-calculus [16, 17], there has been some efforts towards computational models for mobile systems, e.g. based on abstract state machines [15], on Mobile Ambients [3], and on Actors [1]. However, to be used in practical applications, high-level specification languages as well as programming languages whose semantics can be described using such models must be provided. There are some proposals of such programming languages (e.g. KLAIM [7], Pict [20], Nomadic Pict [25]), but on the level of specification there is still no formal method that is largely used for mobile applications.

A formal specification language allows the non ambiguous description of a system, using well-defined syntax and semantics. The formal specification language used in this work, is based on the restricted form of graph grammars called Object-Based Graph Grammars [7]. Besides being a visual language, the object based style adopted is familiar to most developers, making the formalism easier to be understood, used and analyzed, being also closer to implementation languages that follow the object-oriented paradigm. OBGGs were extended in [7] to become a suitable specification technique for mobile code applications. This extension introduced the notions of locality and mobility. Here, we briefly present this formalism.

An application is composed by a set of instances of entities. Roughly speaking, each entity corresponds to a class: it specifies a type of object, its internal state, types of messages it may respond to and send, and the behavior of instances of this entity when receiving a message. Moreover, to be able to simulate and verify each entity separately, the entity has also an interface description, that consists of an abstract description of the behavior of the entity itself (*the export interface*) and an abstract description of the behavior of entities used by this entity (*the import interface*). Each of these three components of an entity (behavior, import and export) is described by an OBGG. The existence of description of behavior in the interfaces makes it possible to check, when building an application by composition of entities, whether the assumed behavior of one entity (import interface) is in accordance with the corresponding actual behavior of the used one (export interface). Due to space limitations, we here we will concentrate on the behavior of each entity.

An OBGG has three components: a type graph, and initial graph and a set of rules. The type graph specifies the kinds of entities, attributes, messages and parameters that will be used in the description of an entity. The initial graph describes the initial state of instances of that entity. The rules specify the behavior of the instances of the entity. The behavior is always triggered by the receipt of a message. The application of rules successively changes the state of the system, starting from the initial state. A rule can be applied whenever the left-side of the rule is a sub-graph of the current system state graph, this is called a match or occurrence of the rule. When applied, the rule brings the system to a new state defined as: Items in the left-side that are not present in the right-side are *deleted*; items in the right-side that are not present in the left-side are *created*; and items that are present in both sides of the rule are *preserved*.

Two (or more) enabled rules are in conflict if their matches need write access to common items. Many rules may be applied in parallel, as long as they do not have write access to the same items of the state (even the same rule may be applied in parallel to itself, using different matches).

To represent mobile applications, two specialized entities were defined: places and mobile components. These entities capture the basic behavior concerning communication and mobility. Places represent the possible locations where mobile components may execute, offering basic functions as storage, communication and computational power, and access to services. Furthermore, places offer message passing and move services to mobile components. Mobile components represent software components that may migrate from place to place during their execution, using resources and services from the places they visit. When creating a mobile application, the user may use the entity place (but he/she is not allowed to modify this entity), and specialize the entity mobile component as he/she wants. This will be illustrated in the following example.

Suppose a *Mobile Component (MC*) that shall visit 3 market *Places (P_1 to P_3)*. In each place, *MC* asks for the name of an *Information Service (IS)* in that place and then asks that IS for the price of a specific product. At the end of the journey, *MC* returns to the origin carrying the information about the place that has the best (smallest) price and informs that to the Customer. Figure 1.a shows a the (partial) specification of this entity. This *MC-TypeGraph* graph specifies the types of the attributes and messages of this entity. For example, an instance of *MC* has 7 attributes: 3 belonging to pre-defined data type sets, 3 of type place and one of type customer entity. The elements below the dashed line indicate imported items, whereas the ones above this line are the internal state and messages treated by this entity. The initial graph *MC-InitGraph* indicates the initial state of the attributes when one instance of this entity is created. The gray elements denote instantiation variables. Some of the rules of the specification of this entity are shown. Again, the ones above the dashed line correspond to the specification of the behavior of this entity, and the ones below to the expected specification of the imported (or used) entities (the export interface is not shown). For example, rule *AskNext* specifies that, when receiving a Price message with a price that is slower that the best recorded price ($r<b$), and having more places to visit ($n!=0$), the *MC* component updates its attributes concerning the value and address of the best price (*bestPrice* and *bestPrLoc*) and asks its location where to go next. Rule *ReqMove* specifies that, when sending a message to an instance of *Place*, *MC* expects to receive a message Continue in return (another rule not shown here specifies the expected message when the movement did not succeed).

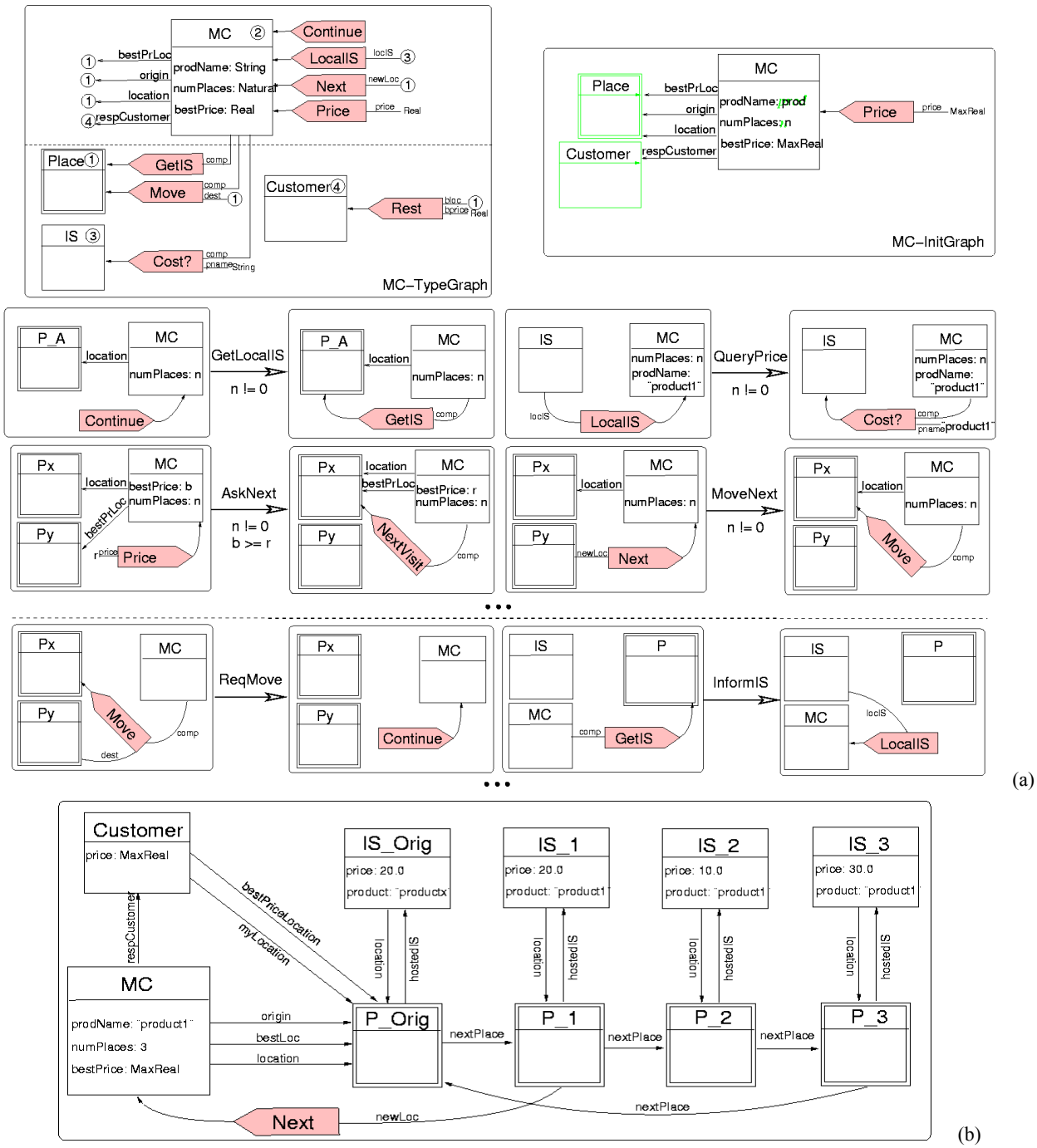

Figure 1. (a) Specification of the MC entity; (b) Model for the Market Application.

Figure 1.b shows an example of model for this application. This is obtained as a composition of the necessary instances of all entities involved. The restrictions put on the kinds of rules, import and export interfaces (see [5]) guarantee that the semantics of the entities is compositional [22, 24], that is, the behavior of each entity is preserved within the whole system. To simulate this model, we just have to use the descriptions of behavior described within each entity (here we do not use the import and export interfaces anymore, they are used just to aid building a consistent model).

3 THE SIMULATION ENVIRONMENT

Simulation models can be very useful as a means of specification and test of a system. They provide a simplified representation of the system, allowing its analysis. The main advantage of using simulation in this work is the possibility of validating a project strategy before implementing the system. Simulating the execution of the system, one can find errors that would be hardly identified when testing the real system execution, specially if one consider open systems with mobile components. Important time and performance aspects can also

be considered without actual implementation of the system. More details on these aspects can be found in Section 3.3.

The simulator developed in the PLATUS project [4], using the Java language [14], allows one to simulate models described in OBGG. This simulator works with entities, which are the system components, corresponding to the OBGG entities, and messages, which are the means of communication between entities. The mapping from OBGGs specifications to simulator entities is straightforward. Graphic tools are under development to allow the graphical creation of simulation models and their automatic conversion to the corresponding simulation code.

3.1 Simulation of OBGGs

In this section we discuss how an object-based graph grammar model as presented in Section 2 can be simulated. For this, we will need some definitions: Given an entity E, a rule r of E and a message m sent to E

- *tmin(m)* is the minimum time in which the message m can be treated (that is, m.tmin+T, where T is the current time);
- *tmax(m)* is analogous *to tmin(m)* for the maximum time;
- *trigger(r)* is the name of the message treated by rule r;
- *readAtr(r)* is the set of (names of) attributes of E that are read-only for rule r (i.e., items that are preserved by this rule);
- *writeAtr(r)* is the set of attributes of E that may be modified by rule r (items that are deleted/created/modified);
- *condition(r)* is a (boolean) condition that must be satisfied by a rule to be applied. Typically, this condition is defined by some requirement on the values of the attributes used by the rule.

Each state of a computation described by a graph grammar is a graph that contains instances of entities (with concrete values for their attributes), messages to be treated, and the current time (simulation entity). In each execution state, several rules may be enabled (and therefore are candidates for execution at that instant in time).

Enabled Rule: A rule *r* is enabled in a state *S* by an instance *m* if *m* is of type *trigger(r)*, *m* belongs to state *S*, each attribute in *readAtr(r)* have the necessary value in S, each attribute in *writeAtr(r)* have the necessary value in *S*, and *condition(r)* is true in *S*. In this case, the pair (*r,m*) is a candidate for rule application.

Rule Application: A pair *(r,m)* can be applied if *r* is enabled by *m*. The effect of this rule application is that all attributes of entity *E* in *writeAtr(r)* will receive the new values described by the rule, message m will be deleted and the new messages described in the right-hand side of the rule will be created. Nothing else in the state is modified by this rule.

Rule applications only have local effects on the state. However, there may be several rules competing to update the same portion of the state. To determine which set of rules will be applied in each time, we need to choose a set of rules that is consistent, that is, in which no two or more rules have write access to the same resources. For this we will use the concept of conflict:

Conflict: A rule application *(r,m)* is in conflict with a rule application set *R* if $writeAtr(r) \cap writeAtr(R) \neq \emptyset$ (where *writeAtr(R)* is the union of all write attributes of rules in *R*).

Note that, in this definition, we do not consider many read accesses with one write access as a conflict. Therefore, these rules may be applied in parallel. This choice is in accordance with the true concurrency semantics model of Graph Grammars [23] and it allows for a high parallelism degree in our system.

3.2 The Simulation Algorithm

Summarizing the definitions until now, an object-based graph-grammar specification can be seen as a set of entities executing in a collaborative way. Each entity keeps its particular state and communicates with others by message passing. The behavior of each entity is described as a set of rules. More than one rule may be candidate to treat one message, but only one will be triggered per message. The selection follows a uniform distributed random function. The execution of a rule may change the values of the entity's attributes, create new entities and generate new messages.

Analyzing the model characteristics we conclude that an object-based graph-grammar model is a discrete-state system: the state of the entities change as rules are executed in response to the messages. The use of discrete-event simulation is thus a natural approach to model evaluation of such systems.

In the literature, many approaches for the execution of discrete-event models were discussed. We may use a fixed-step or event-driven [10] time advancing mechanism and, in an orthogonal view, we may use a single threaded or a multi-threaded (parallel) [12] execution approach. Once parallelism is an intrinsic property of graph-grammars, the multi-thread approach seems to lead to a faithful representation of the behavior of such systems. An interleaving approach, as would be necessary to model parallelism in the single thread approach, would lead to a much more complex implementation, mainly due to the fact that some rule applications that may occur in parallel cannot be sequentialized in any order preserving the same semantics.

In the same way the event-driven time advancing mechanism is a natural choice because the fixed step approach is useful only for a specific set of problems where a fixed step for time advancing is easily identified.

The simulation of discrete events uses a list of messages (or events) sorted by time-stamp, known as event list (EVL). The process of simulation consists in selecting the event (or events) with the smallest time-stamp, executing it (them) and advancing the simulated time. There is only one variable that corresponds to the simulated or virtual time.

For the parallel simulation, the system to be simulated is divided into modules to be executed in parallel. To each module, a logical process (LP) is associated. Each LP can work with the evolution of its own processing time, keeping its own Local Virtual Time (LVT). The set of all LVTs define the Global Virtual Time that represents the advancement of the simulated time. Data exchange between LPs, however, implies the need of synchronization of communication mechanism and clocks [10, 11, 12]. There are two main approaches to achieve this synchronization: the so called conservative approaches and the optimistic approaches. The conservative approaches are based on the idea that an event may only be triggered when it is safe to do it, meaning that the involved LPs are correctly synchronized and, consequently, Local Causality Constrains (LCC) problems are not allowed [10]. Optimistic approaches do not assure that it is safe to execute an event. When an LCC problem occurs, recovery procedures are executed to bring the system into a safe state.

The simulator must ensure that the graph grammars formalism is faithfully simulated. Once the formalism allows that all entities may execute at the same time, a multi-threaded approach seems to be the best choice. Moreover, within an entity, all non-conflicting enabled rules may be applied in parallel. Due to the encapsulation imposed by the object-based modeling, there can be no conflicts among entities.). By analyzing the dependencies among enabled rules, we can construct a set of non-conflicting enabled rules out of an arbitrary set of enabled rules. Messages that cannot be treated in the current time due to such conflicts must be posted again later. If it is not possible to post the message again (because the maximum time would be exceeded), the simulation shall be aborted as there was an specification error: it is not possible to treat the messages within the times they were specified. The proposed algorithm uses a simplified version of a conservative approach adapted to our needs.

Each entity in the model is mapped to a LP. This LP is responsible for the selection of the messages that may be treated in the current time, the choice of the rules that may be activated to treat these messages, and the selection of the set of rules that will effectively be triggered. For each triggered rule, a new LP will be created. This LP has a short life, ending when the execution of the rule ends. This assures a high degree of parallelism among the rules that may be triggered in the same entity (and consequently in all entities) in a specific point in time.

For simplicity, however, we keep only one centralized clock. So all the entities are synchronized by this clock and the degree of parallelism is restricted to those rules that occur at the same time. This is not very efficient if we think about processing time, but is enough to assure the correct simulation of the model and is a good base to improve the algorithm in the future.

Once the clock is unique, we have a kernel. The kernel is a special entity that keeps the EVL and the simulated clock. It is also an LP. When an entity wants to send a message to any other entity (including itself), it must send the message to the kernel. The scheduler will insert the message in the EVL sorted by time stamp. Each time the clock advances, all messages whose $tmin(m) \geq T$, where T is the current time, are sent to their target entities. This way an entity's EVL stores only messages that may be selected for execution in the current time. In each time, all entities will empty their EVLs. The time will advance only when all triggered rules stop to execute.

3.3 Simulation of Mobile Applications

As stated in Section 2, we have represented mobile applications in terms of OBGGs. To do this, specialized entities, namely Place and Mobile Component, were specified using OBGGs. The created entities follow the behavior described in Section 2. Mobile Components are assigned to a Place and may migrate among Places. With these abstractions, mobile applications can be modeled. The developer uses (specializes) these abstractions to represent the specific entities of the application scenario.

Since the simulator supports the simulation of OBGG models, and since mobile applications can be represented using OBGG, it is therefore possible to simulate mobile applications. We have used the simulator to test the behavior of various applications, including the market application presented in Section 2 and currently also a model for active networks.

The tool proved to be valuable since relevant specifications errors could be found during the specification phase, like for instance: sending to an entity a message that is not treated by that entity; wrong definition of conditions of a rule resulting that the rule was never applied, or always applied; and deadlock situations, among others.

Besides the use of simulation as a tool to test specifications, it can also be used to analyze the expected performance of the application. Assigning times to local and remote message delivery as well as to the migration of a software component allows the developer to represent the latency of network links, the amount of time necessary to transfer a mobile component or other time consuming operations, making it possible to compare the performance of different approaches and strategies of distribution and mobility in complex scenarios.

Our simple application scenario can be analysed analytically. Supposing we focus on one aspect to be investigated: to decide which implementation approach should be followed, regarding the best time of response, among the two different approaches to implement the search for best price: (i) using a traditional remote communication means (e.g. remote procedure call - RPC); or (ii) using the mobile agent approach. If we establish costs (delay introduced) for remote and local message passing, as well as for the migration of the mobile components involved, it is straightforward to decide which approach is better. Suppose we have *NShops* shops, one migrating component (that searches the shops sequentially), *NInteract* interactions (messages) in each place, one message to start the search and one message sent back to the customer with the answers, it is straightforward to establish formulas for the overall time of the distributed information retrieval application example (i) using mobility:

$TTM=(NShops \times TM)+(((NShops \times NInteract)+2) \times TLM)$;

and (ii) using remote communications:

$TTRC=(2 \times TLM)+(NShops \times NInteract \times TRM)$,

where *TTM*: Total time with mobility; *TTRC*: total time with remote communications; *TM*: time for migration; *TLM*: time for local message passing; and *TRM*: time for remote message passing.)). Let us assume 10 units of time for *TLM*, 20 for *TRM*, and 60 for *TM*. Figure 2 shows the total times for visiting 3, 6 and 9 shops having 2 to 30 message exchanges in each shop.

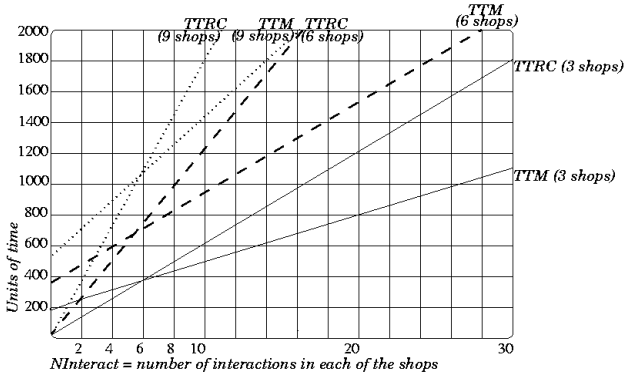

Figure 2. Performance analysis for the Market Application.

However, we can analytically analyse only very simple applications, with low degrees of concurrency, and abstracting from various details. When applications are more complex and we want to analyse their behavior in detail, then it is not trivial to formulate these applications analytically.

To give an example, we have modeled the behavior of active networks [21] with our abstractions. In such scenarios, we have multiple nodes and multiple mobile components running,

various synchronization issues to deal with, and also various aspects which are of interest to be analysed (number of capsules in a node in a given time, average time for a capsule be served in a node, average migration times, capsules lost, etc.). Having the ability to simulate our specifications, easily obtaining also performance measures, makes it possible to investigate more complex situations, in a degree of abstraction that may satisfy our expectations, and turning analysis process much more comfortable to the developer.

4 CODE GENERATION FOR MOBILE APPLICATIONS

Given a formal specification of a system in OBGG, it is essential to have a way of translating this specification into a programming language to be able to generate executable code. This translation, however, is not simple because it requires the maintenance of the system properties defined in the specification.

To accomplish the objective of translating a formal specification into executable code for mobile applications, we started from the translation proposed in PLATUS simulator, discussed in the previous section. According to this translation, entities are described as Java classes. Each entity has and manages a buffer of received messages and creates threads to execute the rules to handle the received messages. Rules are described also by Java classes, where it is defined what message they can handle, what are the conditions that have to be satisfied in order to execute these rules and the transformations in the system state graph performed by their execution.

In a simulated situation, the time, and therefore all events in the system, are ruled by the simulation algorithm. In this process, the kernel plays the important role of passing events (messages) between entities, and of giving a temporal coherence to these events. The goal is to resemble the times of a real situation. In a real situation the events of the system are naturally ordered in time as they occur. Messages can be passed directly from entity to entity as they are generated.

So, our first step to generate a real application out of an OBGG specification is to take the same mapping of OBGGs to Java classes established for simulation, but excluding the simulation time control aspects and excluding the message passing functionality of the simulation kernel, having entities communicating directly. To do this for a distributed scenario, we have adopted a support platform that ensures distribution transparency and FIFO (first-in-first-out) semantics for message delivery, therefore keeping the kernel semantics for message passing. The chosen platform was Voyager from ObjectSpace, Inc [18]. This is a platform totally developed in Java and that provides the necessary support for distributed communication with location transparency, and component mobility with reference resolution. Although we have used this specific platform, we have conceived the modifications in the code to easily allow the implementation to be ported to other support platforms[2].

The behavior discussed above does not consider mobility aspects. The next important aspect to be introduced is the implementation of real entity migration. This can be decomposed in the following sub-aspects:

1. stop the entity in the origin place;
2. a) save internal state as well as b) execution state;
3. transfer it to the destination;
4. resolve references - i.e. the migrating entity continues to be addressed from other entities;
5. resume the entities activities at the destination.

To implement this, we have used a mobility support platform (the same mentioned above for distributed execution) which is able to migrate passive Java objects (without internal activities) between distinct locations where the support platform has been deployed. The platform supports step 2.a (saves only the internal state of the object that represents the entity), 3 and 4. In order to handle with active objects (an object with internal activities - one or more threads), which is the case for entities, we have to build the support for steps 1, 2b and 5. To stop the entity, we have to stop the internal thread (that launches rule processing threads when messages are received) and the threads that are executing rules. That is, we must guarantee that messages will not be processed by the entity, no rules are under execution and no rules will start execution before the move process has finished. When there are no more active threads in the entity, the move process supported by the platform is performed (steps 2.a, 3 and 4). Step 5 was also implemented: when the entity has been migrated to the destination by the platform, the internal thread is restarted. With this, the entity resumes its operation in the point it was stopped, starting to process the messages that were not handled at the former place. Since the platform assures that messages are delivered, without losses, even the destination entity is moving. The behavior above described was implemented in the specialized entities Place and Mobile Component, which use the support platforms functionality. This way, the developer creates his/her mobile application by defining application specific entities which extend the provided basic behavior of Place or Mobile Component as specified in Section 2.

5 CONCLUSIONS AND FUTURE WORKS

The presented work is still in evolution but the created tools (the specification language, the simulator and the code generation) are already a good support for the development of mobile applications. These tools have been enhanced by the realization of tests involving some specific characteristics of this kind of application, like remote communication, synchronization and mobility of entities, and concurrent access to entities.

Some examples of mobile applications were developed to test the code generation proposed. The results obtained in the execution of the generated code are coherent to the simulation results and to the formal specification of the applications. The development of these example applications included the specification of the application in OBGG, the translation of the specification into simulation code, the simulation of the application to test its behavior, the translation of the simulation code into executable code and the execution of the generated code. It is also possible to translate the specification directly into executable code, without translating this into simulation code. The use of simulation is, however, useful in most cases.

The ongoing work considers the development of more complex case studies in order to test and validate the idea of using the formalism, the simulator and the code generation as proposed. Besides this, efforts have been invested to create a tool for the formal verification of specifications in OBGG and to obtain a formal proof that the code generated maintains the graph grammar semantics.

As one could read in this paper, the place and mobile component abstractions define the expected communication and migration characteristics. The definition of these abstractions reflects the expected environment for mobile applications. We

[2] Various platforms are analogous, concerning the support used.

could modify such abstractions to represent other types of environments. One possibility is to represent the existence of failures, like messages that do not arrive in their destinations, places that can be temporarily down, etc. Although this is an open issue, we believe it is possible to use this flexibility to modify the represented environment and to specify wireless environments, where some characteristics like bandwidth, availability and latency could change over the time.

6 REFERENCES

[1] AGHA, G., KIM, W. Actors: A unifying model for parallel and distributed computing. Journal of Systems Architecture 45, 1999, pp. 1263-1277.

[2] ASSIS SILVA, F.M. A Transaction Model based on Mobile Agents. PhD Thesis. Technical University Berlin. FB-Informatik.1999.

[3] CARDELLI, L., GORDON. A., Mobile Ambients - Foundations of Software Science and Computational Structures, LNCS, vol.1378, Springer, 1998, pp. 140-155.

[4] COPSTEIN, B., MÓRA, M. C., RIBEIRO, L. An Environment for Formal Modeling and Simulation of Control Systems. 33rd Annual Simulation Symposium, SCS, 2000. p.74-82.

[5] CORRADINI, A.. Concurrent Computing: From Petri Nets to Graph Grammars. Electronic Notes in Theorical Computer Science 2, Proc. of the SEGRAGRA95 Workshop on Graph Rewriting and Computation, 1995. p. 245-260.

[6] DE NICOLA, R., FERRARI, G., PUGLIESE, R.. KLAIM: a Kernel Language for Agents Interaction and Mobility, Transactions on Software Engineering, vol. 24, nº 5, IEEE Computer Society, 1998. p. 315-330.

[7] DOTTI, F. L., RIBEIRO, L. Specification of Mobile Code Systems Using Graph Grammars. Formal Methods for Open Object-Based Distributed Systems IV, Kluwer Academic Publishers, Stanford, USA, 2000. p. 45-63.

[8] DUARTE, L. M., DOTTI, F. L. Developing Correct Mobile Agent Applications. (portuguese) In: III Workshop de Comunicação Sem Fio e Computação Móvel, 2001, Recife. Anais do III ... Recife: Centro de Informática - Universidade Federal de Pernambuco, 2001. v.1. p.10 - 17. In conjunction with the 3rd IEEE International Conference on Mobile and Wireless Communication Networks (MCWN2001).

[9] FUGGETA, A., PICCO, G. P., VIGNA, G. Understanding Code Mobility. IEEE Transactions on Software Engineering, v. 24, 1998. p.342-361.

[10] FUGIMOTO, Richard. Parallel Discrete Event Simulation. Communications of the ACM, New York, vol. 33, n. 10, October 1990, pp. 31-53.

[11] FUGIMOTO, Richard, NICO, David. State of Art in Parallel Simulation. In Winter Simulation Conference (WSC´92), Arlington, Proceedings..., SCS, 1992, pp. 246-254.

[12] FUGIMOTO, Richard. Parallel and Distributed Simulation Systems.NY, John Wiley & Sons, 2000.

[13] GOLDSZMIDT, G., YEMINI, Y.. Distributed Management by Delegation. In Proceedings of 15th Int. Conf. On Distributed Computing, 1995.

[14] GOSLING, J., MCGILTON, H. The Java Language Environment - A White Paper. SunMicrosystems, 1996.

[15] MAIA, M., BIGONHA, R. Interaction based semantics for mobile objects, In Proc. of the III Brazilian Symposium on Programming Languages, 1999.

[16] MILNER, R., PARROW J. A calculus for mobile processes I, Information and Computation, vol. 100, 1992, p. 1-40.

[17] MILNER, R, PARROW J., WALKER, D. A calculus for mobile processes II, Information and Computation, vol. 100, 1992, pp. 41-77.

[18] OBJECTSPACE. Voyager ORB 4.0 Developer Guide. Objectpace, Inc. 2000.

[19] PERDIKEAS, M., CHATZIPAPADOULOS, F., VENIERIS, I. and MARINO, G. Mobile Agent Standards and Available Platforms, Computer Networks, vol. 31, 1999, pp. 1999-2016.

[20] PIERCE, B. and TURNER, D. Pict: a programming language based on the pi-calculus, Tech. Report 476, Indiana University, 1997.

[21] PSOUNIS, K.. Active Networks: Applications, Security, Safety and Architectures. IEEE Communications Surveys, 1999.

[22] RIBEIRO, Leila, COPSTEIN, Bernardo. Compositional construction of simulation models using graph grammars. Lecture Notes in Computer Science, v.1779, p.87-94, 2000.

[23] RIBEIRO, L.. Parallel Composition and Unfolding Semantics of Graph Grammars. Ph.D. thesis, Technical University of Berlin, Germany, 1996.

[24] RIBEIRO, Leila. Parallel composition of graph grammars. Applied Categorical Structures, v.7, n.4, p.405-430, 1999.

[25] WOJCIECHOWSKI, P., SEWELL, P. Nomadic Pict: language and infrastructure design for mobile agents, In Proc. of the ASA/MA99, 1999.