Electronic Notes in Theoretical Computer Science 50 No. 3 (2001) – Proc. GT-VMT 2001
URL: `http://www.elsevier.nl/locate/entcs/volume50.html` 9 pages

A Graph Transformation View on the Specification of Applications using Mobile Code[⋆]

Andrea Corradini[1]

Dipartimento di Informatica, Università di Pisa, Pisa, Italy

Fernando Luís Dotti[2]

Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Brazil

Leila Ribeiro[3]

Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brazil

Abstract

The main aim of this extended abstract is to discuss the requirements of a specification method for mobile code applications and analyze to what extent Graph Transformation Systems can be used to meet these requirements. We suggest some extensions to the theory of Graph Transformation which seem to be desirable to cope with this kind of applications.

1 Introduction

Highly distributed networks (e.g. Internet) have now become a common platform for large scale distributed programming. These environments are often called *open environments*, being characterized by: massive geographical distribution; high dynamicity (appearance of new nodes and services); no global

⋆ Research partially supported by project ForMOS (Métodos Formais para Código Móvel em Sistemas Abertos) - grant CNPq 520269/98-5, by the Research Training Network GETGRATS (General Theory of Graph Transformation Systems), and by the MURST project *Teoria della Concorrenza, Linguaggi di Ordine Superiore e Strutture di Tipi (TOSCA).*

[1] Email: `andrea@di.unipi.it`

[2] Email: `fldotti@inf.pucrs.br`

[3] Email: `leila@inf.ufrgs.br`

control; partial failures; lack of security and high heterogeneity due to the diversity of communication links (delay, throughput), cooperating organizations, services offered, etc.

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* [11] has received special attention due to its flexibility and potential use in various application fields, like network management [11], electronic commerce [19], distributed information retrieval [16], advanced telecommunication services and active networks [27], active documents, workflow management systems, and disconnected operations (namely, the ability to launch remote computations, switch off the local node (e.g., a laptop), switch on the node later, and receive the results of the remote computations).

Currently there are standards, platforms and languages available for mobile code [22]. Java is being widely used due to high portability and dynamic binding, among other features. However, there are still problems to be addressed in order to build a sound support for mobile code. One of them is the lack of formal basis: the ideas around mobile code and active networks and their implementations emerged from a practical approach. An abstract semantic framework (including methods for specification, verification and analysis) to formalize the model of computation of Internet applications is clearly needed, and missing. Such semantic framework may provide the formal basis to discuss and motivate controversial design/implementation issues and to state and certify properties in a rigorous way.

Graph Transformation Systems are used as specification formalism as well as to describe models of computation of concurrent and distributed systems [24,9,10]. In this extended abstract we will discuss how far existing concepts in the area of graph transformations may be used to describe mobile code systems in open environments, and give ideas of suitable extensions.

2 Mobile Code Applications and Open Environments

In the typical environment for mobile code applications the distributed environment comprises a set of places and a set of mobile components:

Places represent the needed infrastructure to support mobile applications. This infrastructure comprises hardware and software (middleware) support to offer locations where mobile components can run, move into and leave. Beyond these basic functions, places can be configured to support additional functionalities as needed. While building mobile applications, the developer assumes that places offer a given level of functionality. The lowest level is the support to mobility and to communication among components in the same place.

Mobile components are software components that, during their execution, may migrate from place to place to use other components as well as basic facilities like processing power, storage and communications. A mobile component has internal data or state, code, and a set of meta-data or attributes (e.g. identifier, credentials, originator, operational status, etc.). *Mobile applications* can be build from various mobile components, which may run concurrently and cooperate.
While developing mobile code applications for open environments, the following main characteristics have to be taken into account:

Open/Dynamic environment: Places must offer a basic functionality level and be easily configurable in order to better fit the goals of the distributed infrastructure. In many situations, places must be highly customizable in order to support different applications in a distributed environment. It is important to easily install components in various places and use them afterwards (e.g. for active networks).

The distributed environment considered is characterized by being open. Software components and places are independent in the sense that they have distinct ownership and therefore distinct authorities to decide upon their management. In such an environment, these entitites have distinct life-cycles, and no global state can be provided/obtained. Nonetheless, entities often cooperate in order to achieve their goals.

No location transparency: Components are designed without location transparency. Migration is not transparent, but rather specified explicitly.

Autonomy/Concurrency: Components may create other mobile components and run concurrently in the same or different places. Due to the autonomy of each component, it is important to design/represent each component independently.

Modularity/Cooperation: It is important to have well defined component interfaces and to be able to compose various components to build applications.

Failures: While designing a distributed application the designer has to assume some behavior of the environment where the application is to run. This has already been stated for places. Another very important aspect is to consider the failure behavior of the environment. Therefore it is desired that the designer may state the expected failure behavior of the environment and build its application to work on it.

3 A Graph Transformation View

Graph Transformation Systems allow to represent the state of a distributed system as a graph, and to model the computations of such systems via local applications of rewriting rules. The explicit representation of the topology makes this formalism particularly suitable for specifying mobile code applications, because essential information does not need to be coded, but can be handled in a direct and intuitive way.

In this section we will consider each of the characteristics of mobile code applications listed in the previous section in turn, and for each of them we will discuss how far Graph Transformation Systems are adequate to model it, and possibly which extensions would be desirable.

No location transparency: The specification of places may be implicit or explicit. For instance, implicit representation of places is used in the π-calculus [20,21] and mobile ambients [2]. This makes the model more independent from implementation, but also makes it more difficult to realize a specification. An explicit representation of places and component identities seems closer to current implementation platforms.

Using Graph Transformation Systems, it comes natural to model places and components as nodes, and the relations modelling spatial distribution (like adjacency of places, presence of a component on a place, ...) as edges. Actually, the possibility of having an explicit representation of the system's topology is a strong point in favour of this specification formalism.

Autonomy/Concurrency: Typically, a mobile code application is specified in terms of various components, which act autonomously and concurrently, cooperating to accomplish some task. The components usually interact among themselves via message passing, and can create new components during execution.

Graph Transformation Systems allow one to specify the evolution of distributed systems via rules, which have a local effect when applied. Concurrency is to a large extent built-in, as independent local modifications can be performed in any order without affecting the result [1]. There is no automatic modelling of message passing, but there are various reasonable proposals in the literature about modelling message-passing based systems using graph transformations [15,17,23,4,5]

Modularity/Cooperation: The complete specification of a mobile code application requires the specification of the various components as well as the specification of the middleware (the places). In realistic situations, monolithic specifications are difficult to manage and mantain. Typically, the middleware can be specified separately, and the specification reused for many systems. Also the various components of a system can be specified independently by different people/groups. Therefore, modularization concepts have to be supported by the specification formalism. A module must offer an export interface which provides at least an abstract description of the behaviour of the component it implements. In practical applications, usually this is given by the signature of the functions (methods) offered to other components [18].

Many approaches to the modular design of Graph Transformation Systems exists, see [8] for an informal comparison of some of them. In all such approaches a module includes at least a description of the class of graphs handled by the modules (for example, via a *type graph*) and a collection of graph transformation rules. Concerning the export interfaces, some of the approaches (like GRACE [14] and PROGRES [25]) only allow to export procedure names, while others (like DIEGO [26] and TGTS [12]) also allow to export rules. This feature could be used to describe abstractly the behavior of the component in terms of kinds of observable events (for example, messages received/sent by a component). This would allow for a rely–guarantee approach [6].

Like in all approaches to the specification of complex systems, it is desirable for the specification formalism to have a formal semantics, which can be used to verify the specification and to validate it with respect to the requirements of the mobile code application. Such a semantics should be compositional: typically the semantics of the whole system should be obtained as a suitable composition of the semantics of the modules constituting the system itself. In an ideal situation, the export interfaces of modules would also include semantical information about the exported rules/procedures, and the formalism would support a notion of *correctness* of a module, assuring that the abstract semantics described in the interface is in accordance to the one described by the rules of the specification.

Open/Dynamic Environment: Decomposing the application into different modules is not yet enough to specify mobile applications because the environment in which the components will execute is highly dynamic (places and other components may be created/deleted at any time), implying that the specification of a mobile application can not follow the "closed world assumption". As a consequence an open, loose semantics is needed for this kind of application. The semantical framework proposed in [13] seem to satisfy most of these requirements.

Dynamic environment: In many situations, it is interesting to allow the creation of multiple instances of the same component. For example, if there is a list of places to visit asking for some information, one could create one mobile component to visit each of the places and bring the information back. The only difference among them is the place they have to visit. Some approaches to modularity in graph transformation allow one to specify formal parameters for modules. For example, in both PROGRES and CGSPEC [7] the procedures exported by a module may have parameters that can bound by the client at instantiation time.

Components may die at any moment, even if other components have pointers to them (for example, know their names). The middleware is responsible for handling these situations: it warns a component in case it tries to communicate with a non-existing component. Also, the middleware may support distributed garbage collection. In this case, a component does not die but may stop its internal activity and remains still answering to messages from other components. When the other components cease to refer to that component, it may then be erased by the distributed garbage collection.

In a graph transformation approach, a possible way to handle deletion of components is to mark them as dead and provide (exception) rules for handling the cases when a component tries to communicate with them. These rules, as well as those for distributed garbage collection, if any, should be part of the specification of the middleware, as this is a service provided by this level. It would be desirable to have exceptions handled by the semantics in a clean way.

Failures: Mobile code applications have to cope with the unreliability of the open environment. Typically, one should provide levels of confidence in the implementation of this systems. For example, one may want to ensure that provided nothing fails, the system behaves as expected; or that if some servers (these must be specified) fail, the system still behaves correctly, etc. Sophisticated analysis techniques would be needed for addressing such issues.

An approach to the static analysis of graph trasformation systems is proposed in [3], where using a modified unfolding construction, from a given system a finite graph can be extracted together with an underlying Petri net. All graphs reachable from the initial graph of the system can be mapped homomorphically to the extracted graph, which can therefore be used to check, for all reachable graphs, suitable properties which are reflected by morphisms (like the non-existence of a path or of a cycle). Furthermore, the underlying net summarizes some causal dependency relations among the rules of the system, and can be analyzed using standard Petri net techniques, providing additional information about the original system. The application of this technique to mobile code applications is topic of future research.

4 Concluding Remarks

In the previous sections we informally discussed some characterizing aspects of mobile code applications, which should be addressed adequately by specification formalisms aimed at describing such systems. We argued that Graph Transformation Systems are a good candidate for this goal, explaining the way they support in a direct way some of the required features. For other aspects, even if various proposals in the literature address them under various perspectives, certainly more work is needed.

In the spirit of the workshop for which this extended abstract is written, the present contribution is mainly methodological. The research direction that we intend to pursue consists in selecting a specific approach to graph transformation which satsfies as much as possible the properties singled out in the previous section, and to experiment with it the specification of sample mobile code applications.

A first attempt in this direction is reported in [4,5], where a restricted form of graph grammar is used to model code mobility. Components, places and messages are modeled as nodes. The behavior of places is specified by rules which realize typical middleware functionalities, like forwarding messages to agents located on a remote place, or handling the migration of agents. The behavior of each component is specified in a reactive way: each rule must delete a message vertex, meaning that this message triggers the application of this rule. No true module concept is used in the specification, but there are some further restrictions to rules to assure encapsulation properties of components.

References

[1] Baldan, P., Corradini, A., Ehrig, H., Löwe, M., Montanari, U. and Rossi, F., *Concurrent Semantics of Algebraic Graph Transformations*, in [10].

[2] Cardelli, L. and Gordon, A., *Mobile ambients,* Foundations of Software Science and Computational Structures, Lecture Notes in Computer Science, vol. 1378, Springer, 1998, pp. 140–155.

[3] Baldan, P., Corradini, A. and König, B., *A Static Analysis Technique for Graph Transformation Systems*, submitted for publication, 2001.

[4] Dotti, F. and Ribeiro, L., *Specification of mobile code systems using graph grammars*, Proc. FMOODS'00 Formal Methods for Open Object-based Systems, Stanford (USA), Ed. Kluwer, 2000 (to appear).

[5] Dotti, F. and Ribeiro, L., *Code mobility in open systems: a formal approach*, Proc. PDPTA'00 Parallel and Distributed Processing Techniques and Applications, Las Vegas (USA), 2000.

[6] Duarte, C.H.C. and Maibaum, T., *A rely-guarantee discipline for open distributed systems design*, Information Processing Letters, vol. 74, 2000, pp. 55–63.

[7] Ehrig, H. and Engels, G., *Pragmatic and semantic aspects of a module concept for graph transformation systems*, in Proceedings Fifth Int. Workshop on Graph Grammars and Their Application to Computer Science, Lecture Notes in Computer Science, vol. 1073, Springer, 1996, pp. 137–154.

[8] Heckel, R., Engels, G., Ehrig, H. and Taentzer, G., *Classification and comparison of modularity concepts for graph transformation systems.* In [9], pp. 669–690.

[9] Ehrig, H., Engels, G., Kreowski, H.-J. and Rozenberg, G. (editors), *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools.* World Scientific, 1999.

[10] Ehrig, H., Kreowski, H.-J., Montanari, U. and Rozenberg, G. (editors), *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency, Parallellism, and Distribution.* World Scientific, 1999.

[11] Fuggeta, A., Picco, G. and Vigna, G., *Understanding Code Mobility*, Trans. On Software Engineering, IEEE, vol. 24, 1998, pp. 342–361.

[12] Große-Rhode, M., Parisi Presicce, F. and Simeoni, M., *Refinements and modules for typed graph transformation systems*, in Proceedings WADT'98, Lecture Notes in Computer Science, vol. 1589, Springer, 1999, pp. 137–151.

[13] Heckel, R., *Open Graph Transformation Systems: A New Approach to the Compositional Modelling of Concurrent and Reactive System*, Ph.D. Thesis, Technische Universität Berlin, 1998.

[14] Heckel, R., Hoffmann, B., Knirsch, P. and Kuske, S., *Simple Modules for GRACE*, in Proceedings TAGT'98, Lecture Notes in Computer Science, vol. 1764, Springer, 2000, pp. 383–395.

[15] Janssens, D. and Rozenberg, G., *Actor Grammars*, Mathematical Systems Theory, vol. 22, 1989, 75–107.

[16] Knudsen, P., *Comparing Two Distributed Computing Paradigms - A Performance Case Study*, M. Sc. thesis, University of Troms, 1995.

[17] Korff, M., *Generalized graph structures with application to concurrent object-oriented systems*, Ph.D. thesis, Technical University of Berlin, 1995.

[18] Lange, D.B. and Chang., D.T., *Java Aglet Application Programming Interface (J-AAPI) White Paper.* IBM Corporation. November, 1997, `http://www.trl.ibm.co.jp/aglets/JAAPI-whitepaper.html`.

[19] Merz, M. and Lamersdorf, W., *Agents, Services and Electronic Markets: How Do They Integrate?*, In Proc. of the International Conference On Distributed Platforms, IFIP/IEEE, 1996.

[20] Milner, R. and Parrow, J., *A calculus for mobile processes I,* Information and Computation, vol. 100, 1992, pp. 1–40.

[21] Milner, R., Parrow, J. and Walker, D., *A calculus for mobile processes II,* Information and Computation, vol. 100, 1992, pp. 41–77.

[22] Perdikeas, M., Chatzipapadopoulos, F., Venieris, I. and Marino, G., *Mobile Agent Standards and Available Platforms.*, Computer Networks, vol. 31, 1999, pp. 1999–2016.

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

[24] Rozenberg, G. (editor), *The Handbook of Graph Grammars, vol. 1: Foundations*, World Scientific, 1997.

[25] Schürr, A. and Winter, A.J., *UML Packages for PROgrammed Graph REwriting Systems*, in Proceedings TAGT'98, Lecture Notes in Computer Science, vol. 1764, Springer, 2000, pp. 396–409.

[26] Taentzer, G. and Schürr, A., *DIEGO, Another Step Towards a Module Concept for Graph Transformation Systems*, in Proc. of SEGRAGRA'95 "Graph Rewriting and Computation", Electronic Notes of TCS, 2, 1995, `http://www.elsevier.nl/locate/entcs`.

[27] Tennenthouse, D. L., Wetherall, D. J., Smith, J. M., Minden, G. J. and Shiaskie, W., *A Survey of Active Network Research*, IEEE Communication Magazine, January 1997.