Formal Foundations for Compositional Software Architectures

José Meseguer and Carolyn Talcott

meseguer@csl.sri.com, clt@cs.stanford.edu

Position Paper
OMG-DARPA-MCC WORKSHOP ON
COMPOSITIONAL SOFTWARE ARCHITECTURES

1 Introduction

The current, and quite real, technical problems blocking software compositionality in practice could benefit from formal foundations providing clear semantics, new architecture and software composition methods, and better languages. The choice of foundations is however a delicate matter, because the proposed semantic framework should explictly address and support key features such as openness, distribution, object-orientation, compositionality, and mobility.

Rewriting logic is a promising semantic framework supporting such key features and offering simple and rigorous formal foundations for software composition and evolution. It also provides a wide-spectrum approach in which architectural descriptions, executable formal specifications, and distributed and mobile code can be related, composed, transformed, and integrated in a rigorous way. Reflection is a key feature of rewriting logic of great usefulness for software composition, dynamicity, and mobility.

2 Semantic Framework

The choice of a semantic framework to provide formal foundations for compositional software architectures is a matter of utmost importance. It is crucial to insure that the framework is sufficiently flexible so that it can adequately represent and express the key relevant features such as: openness; distribution; object orientation; diverse interaction models; mobility; and composition and coordination mechanisms.

For example if a framework is founded on higher-order functions it is difficult to model naturally concurrency, non-determinism, and interaction. Similarly if a framework has a particular model of concurrency built in, such as process algebras or Petri nets, then it will be difficult to express naturally other models of interaction.

Rewriting Logic seems like a promising candidate for a foundational framework for compositional software architectures. For example it has successfully been used as a framework to unify a very wide range of models of concurrency [11, 13]. In particular many different styles of concurrent objects open software components have been formally modelled in rewriting logic [12, 16, 14]. It has also been used as a general logical framework within which logics of very different natures have been naturally and concisely represented [8, 9]. More recently it has been used to demonstrate semantic interoperation of different architectural styles [10]. Furthermore rewriting logic is a wide-spectrum framework that supports semantics preserving transformations from high-level specifications to executable prototypes and from these to efficient parallel and distributed code [7]. There are currently three language implementations that support execution of rewriting logic specifications [2, 6, 1], including the Maude language on wich we are currently working [2, 12].

Rewriting logic is a logic of concurrent action and change in which the elementary actions of a system are axiomatized as rewrite rules. These rules describe local changes in terms of algebraically specified patterns. An important aspect of rewriting logic is that it is *reflective*. That is, it can represent and act upon its own meta-level descriptions [3, 4]. Reflection supports dynamic evolution and adaptation of components and of system architectures, hierarchical and distributed management of runtime services [17], powerful and extensible module composition operations, and development of formally based analysis and reasoning tools [5].

2.1 Interoperability

Our approach to formal interoperability of software components and architectures is to represent the underlying interaction models and architectural styles as classes of rewriting logic theories meeting specific structural constraints that reflect and enforce the features of these models. Composition within a given style is then straightforward at the operational level, as the algebra of system configurations lifts naturally to an algebra of computations. To understand how to provide compositionality at more abstract levels we have developed a series of composable models for actor systems (distributed objects that interact by asynchronous communication) ranging from the rewriting based operational model to abstract trace like models [15]. Again the key is being able to lift the algebra of configurations to an algebra of models at each level of abstraction. To compose components using different interaction models we define a small number of adaptor or coordinator components that provide the additional glue needed for meaningful interaction. A next step in building the formal foundations is to extend our methods to develop composable models at more abstract levels for mixed models of interaction. Usings these basic adaptors, one can then build complex connectors for coordinating the interactions of components and have application-independent characterizations at multiple levels of abstraction of the coordination behavior that compose with component specifications to give overall system functionality.

2.2 Reflection

Reflection plays a role in formal foundations in two forms: logical and computational. Logical reflection provides modularity and extensibility of the logical framework both in terms of expressive power and in support of building formally based tools.

A key consequence of logical reflection is that modules—corresponding to rewrite theories—become directly accessible as first-class *data*, so that very powerful module composition and module transformation operations can be rigorously defined and executed in the logic. Another important and related consequence is that reflection can be exploited to efficiently implement representations from other logics into rewriting logic, and in particular to build *formally-based tools* for such logics with much greater ease and extensibility than in conventional ways. Our experience building an inductive theorem prover and a Knuth-Bendix equational completion tool this way fully support this conclusion.

Computational reflection provides extensibility of the computational model and flexibility in building robust and adaptable systems. Formalizing computational reflection is supported in rewriting logic at two levels: direct formalization of the meta level of the computational model; and extension of meta-level rewriting theories to describe and manipulate the classes of theories corresponding to the computational model.

2.3 Dynamicity and Mobility

Reflection is also a very powerful technique to support dynamicity of software architectures and component configurations, and to design mobile languages and systems. Dynamic architectures are of great importance in a distributed computing environment, and are crucial for evolving perpetual systems that cannot be shut down. The dynamic transformation and adaptation of components and their configurations can be naturally expressed by means of reflection.

Mobility involves moving around not only data, but also software modules. By reflection, such modules become also data that can be moved and can be manipulated as part of a mobile computation. Therefore, both the semantics and the language design of a mobile language can greatly benefit from an explicitly reflective framework that at the same time supports concurrency and distribution such as rewriting logic.

3 Some Promising Future Directions

We are currently designing Mobile Maude, a rewriting logic-based mobile language with features for both mobility and reflection. The declarative nature of the language, combined with the formal analysis techniques and tools that we are developing, will make the verification of safety-critical properties in mobile programs much easier than for conventional mobile languages.

Using Maude to develop additional formal analysis and reasoning tools is very promising, both because experience indicates that this is an efficient means of developing such tools, and because tools that work within the rewriting logic framework have a natural basis for interoperation and combination.

Computational reflection provides a mechanism for separate consideration of system-wide properties (the ilities), and component functionality, the former being achieved by meta level programming. These mechanisms give rise to a different dimension of compositionality that characterizes the interactions between objects or components and their meta-level descriptions. An important open problem is to understand the semantics of these interactions and to develop algebras describing at suitable levels of abstraction the installation of meta-level controllers and managers in a configuration. These would allow independent specification of policies for coordination, dependability, security, etc. and application functional behavior.

References

[1] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and M. Vittek. ELAN: A logical framework based on computational systems. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996. `http://www1.elsevier.nl/mcs/tcs/pc/volume4.htm`.

[2] Manuel G. Clavel, Steven Eker, Patrick Lincoln, and José Meseguer. Principles of Maude. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996. `http://www1.elsevier.nl/mcs/tcs/pc/volume4.htm`.

[3] Manuel G. Clavel and José Meseguer. Axiomatizing reflective logics and languages. In Gregor Kiczales, editor, *Proceedings of Reflection'96, San Francisco, California, April 1996*, pages 263–288. Xerox PARC, 1996.

[4] Manuel G. Clavel and José Meseguer. Reflection and strategies in rewriting logic. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996. `http://www1.elsevier.nl/mcs/tcs/pc/volume4.htm`.

[5] Manuel G. Clavel and José Meseguer. Internal strategies in a reflective logic. In B. Gramlich and H. Kirchner, editors, *Proceedings of the CADE-14 Workshop on Strategies in Automated Deduction (Townsville, Australia, July 1997)*, pages 1–12, 1997.

[6] K. Futatsugi and T. Sawada. Cafe as an extensible specification environment. In *Proc. of the Kunming International CASE Symposium, Kunming, China, November*, 1994.

[7] Patrick Lincoln, Narciso Martí-Oliet, and José Meseguer. Specification, transformation, and programming of concurrent systems in rewriting logic. In G.E. Blelloch, K.M. Chandy, and S. Jagannathan, editors, *Specification of Parallel Algorithms*, pages 309–339. DIMACS Series, Vol. 18, American Mathematical Society, 1994.

[8] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. Technical Report SRI-CSL-93-05, SRI International, Computer Science Laboratory, August 1993. To appear in D. Gabbay, ed., *Handbook of Philosophical Logic*, Kluwer Academic Publishers.

[9] Narciso Martí-Oliet and José Meseguer. General logics and logical frameworks. In D. Gabbay, editor, *What is a Logical System?*, pages 355–392. Oxford University Press, 1994.

[10] J. Meseguer and C. Talcott, 1997. Technical presentations to EDCS Architecture Cluster Meeting, April and July 1997.

[11] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[12] José Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In Gul Agha, Peter Wegner, and Akinori Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, pages 314–390. MIT Press, 1993.

[13] José Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In *Proc. CONCUR'96, Pisa, August 1996*, pages 331–372. Springer LNCS 1119, 1996.

[14] C. L. Talcott. An actor rewriting theory. In J. Meseguer, editor, *Proc. 1st Intl. Workshop on Rewriting Logic and its Applications*, number 4 in Electronic Notes in Theoretical Computer Science. North Holland, 1996.

[15] C. L. Talcott. Composable semantic models for actor theories. In T. Ito M. Abadi, editor, *Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, page to appear. Springer-Verlag, 1997.

[16] C. L. Talcott. Interaction semantics for components of distributed systems. In E. Najm and J-B. Stefani, editors, *Formal Methods for Open Object-based Distributed Systems*, pages 154–169. Chapmn & Hall, 1997.

[17] N. Venkatasubramanian and C. L. Talcott. Reasoning about Meta Level Activities in Open Distributed Systems. In *Principles of Distributed Computation*, 1995.