Rewriting Logic and Secure Mobility

José Meseguer and Carolyn Talcott
SRI International and Stanford University

Why Rewriting Logic

The purpose of this note is to propose a new approach to the formal semantics and declarative programming of mobile applications that meets the needs of mobile computing in a particularly natural and simple way. This approach also offers a good formal basis for defining the new security models that mobile computing so clearly needs, and formal methods for verifying that declarative mobile programs satisfy key security properties.

Our key proposal is adopting rewriting logic [12] as a semantic foundation for mobile computing and for the new security models that mobile computing requires. A related proposal is using rewriting logic not only as a formal foundation, but also as a mobile programming language. In particular, the rewriting logic language Maude [13, 3], appropriately extended with constructs for mobility, can provide a very convenient declarative style of programming mobile applications.

The main reason why rewriting logic seems to us a good semantic framework for mobile computing is the naturalness and simplicity with which it can express concurrent and distributed computation. There is already extensive evidence that this is the case for many concurrency models [12, 14]. For mobile computing this can be articulated in more detail by pointing out that, in particular, rewriting logic supports very well concurrent object models—such as the actor model and other variants [20, 13]—that seem very appealing and natural in conceptualizing mobile computations as collections of active agents that move around doing things for us and communicate with each other in appropriate ways. A further important property of rewriting logic is its reflective character [4]. We view reflection as a key feature of mobile computing, because mobile computations typically involve different computations occurring at possibly different sites that need good metalevel ways of communicating with each other, and also because of the modular and composable way in which issues like security and fault tolerance can be approached with reflective techniques.

Reflection and security models are further discussed in subsequent sections. We sketch here an impressionistic account of how rewriting logic models concurrent systems. Rewriting logic is a logic to reason correctly about the evolution in time of a concurrent system. The distributed state of a concurrent system is represented as a *term* whose subterms represent the different components of the concurrent state. Typically, however, the *structure* of the concurrent state may have a variety of equivalent term representations because it satisfies certain *structural laws*. For example, in a concurrent object-oriented system the concurrent state, which is usually called a *configuration*, has typically the structure of a *multiset* made up of objects and messages. Therefore, we can view configurations as built up by a binary multiset union operator that satisfies structural laws of associativity and commutativity. This means that the way in which we parenthesize the multiset or arrange the order of its elements is immaterial: all these ways represent the same multiset.

As a consequence, we can abstractly represent the configuration of a typical concurrent object-oriented system as an equivalence class $[t]$ modulo the structural laws of associativity and commutativity obeyed by the multiset union operator of a term expressing a union of objects and messages, i.e., as a multiset of objects and messages.

In rewriting logic, sentences are rewrite rules of the form

$$[t] \longrightarrow [t']$$

What those sentences axiomatize are the basic *local transitions* that are possible in a concurrent system. For example, in a concurrent object-oriented system the rewrite rules may specify the local transitions by which objects react to the reception of a message changing their state and possibly sending other messages and creating some new objects.

The system then evolves concurrently by the process of concurrent rewriting, that is of concurrently performing such local transitions *modulo* the structural laws satisfied by the system. For example, thanks to the structural laws of associativity and commutativity for multiset union, a concurrent configuration of objects and messages behaves as a very fluid "soup" in which objects and messages sent to them can come into contact with each other, and can then react according to the local transitions specified by the rules.

What the rules of deduction of rewriting logic support is sound and complete reasoning about the complex concurrent transitions that are possible in a concurrent system whose basic local transitions are axiomatized by given rewrite rules. That is, the sentence $[t] \longrightarrow [t']$ is provable in the logic using the rewrite rules that axiomatize the system as axioms if and only if the concurrent transition $[t] \longrightarrow [t']$ is possible in the system. A precise account of the model theory of rewriting logic fully consistent with the above system-oriented interpretation, and proving soundness, completeness, and the existence of initial models is given in [12].

There is an extensive body of research in rewriting logic, involving more than sixty papers by authors in the US, Europe and Japan, and several language implementations; see [6] and the references in [14].

Reflective Semantics

Reflection, that is the capacity for a computational system to represent and to act upon key aspects of its own computation, is very relevant for mobile computing. Indeed, it is not just state but actual programs that have to be moved around; and, in addition, a mobile computation typically involves multiple computational sites executing different code and somehow coordinating their activities.

All this is probably quite obvious and uncontroversial. The problem is that reasoning formally about reflective systems is usually harder than reasoning about non-reflective ones; and that the difficulties are aggravated by the fact that some reflective languages lack a formal semantics. A key advantage of rewriting logic in this regard is that it is reflective in a precise mathematical sense, and it is therefore possible to reason formally about its reflective capabilities inside rewriting logic itself [4].

The key idea about how rewriting logic is reflective is that there is a *universal theory* U that can simulate all other finitary theories in the logic, including itself [5, 4]. Specifically, a theory T can prove a sentence φ if and only if U can prove a sentence representing that T proves φ.

The representation process is very important, because it allows bringing down to the "object level" meta-level entities such as theories. That is, a theory T is represented by an expression $\overline{T}$ in the object level of U. In this way, modules in a language like Maude that are theories in rewriting logic become first-class computational entities in the logic and this endows Maude with powerful reflective and meta-programming capabilities [4, 3]. Furthermore, given the great flexibility of rewriting logic as a semantic framework that naturally expresses a wide range of models of computation [14], it also becomes possible to give within it a formal semantics for other reflective computational models. In particular, thanks to the natural way in which rewriting logic supports object-oriented concurrent models such as actors [13, 20], one can support in this way a variety of reflective actor models as those discussed below.

For mobile code, rewriting logic reflection provides a form of higher-order abstraction, because code can be reified and can be shipped around as data—in Maude, as the term $\overline{T}$ representing the theory T of a module—just as any other expression, and can then be reflected back to runnable code in the particular location where it has to run. In fact, this is a very practical way of achieving mobility in a rewriting logic language like Maude, because Maude programs are rewriting theories that are typically much more compact than corresponding programs written in imperative languages.

Other important consequences of reflection are *modularity*, *composability*, and *distributed hierarchical metalevel control*, that can be supported by rewriting logic based reflective actor models. In particular, one can support in this way two reflective architectures for actor computation that have been used to provide a basis for defining and reasoning about dynamic adaptable distributed systems: the *onion skin* and the *two level actor model*.

The *onion skin* or layered architecture aims at modular specification of interaction policies for components of distributed systems. In this model each actor has a single meta-actor that can observe and modify its behavior. Since meta-actors are themselves actors this process can be repeated indefinitely. The onion skin model provides a simple means of structuring interaction policies in layers and allows for policy enforcers to cleanly disappear when no longer needed. In this way, different protocols for security, reliability, quality of service and other policies can be composed and modified dynamically as the system runs and new needs arise by adding new meta-levels. Several prototype language implementations have been developed based on these ideas [2, 8, 19, 7, 18, 1].

The *two level actor model* (TLAM) provides mechanisms for describing distributed system services built from services local to a node. Here each node has a group of base-level (application) actors and a group of meta-level actors that observe and control the base-level behavior. Meta-level actors from different nodes in a network interact to provide distributed systemwide services. This model has been used to formalize and reason about a distributed garbage collection algorithm [21]. It has also been used to specify and reason about the safe composition of system-level activities such as remote creation, migration, and recording of global snapshots of system properties such as accessibility relation between actors [22, 23].

Rewriting Logic and Secure Mobility

Security is probably the biggest technical challenge that needs to be solved to achieve widespread acceptance of mobile computing. For example, there has been much interest and discussion in the WWW community related to the security of Java applets [11, 16, 15]. The designers of Java and the Java Virtual Machine have made substantial efforts to provide powerful executable content in HTML files while protecting security of the visitors to these files. However, serious problems remain. To quote MacGraw and Felten([11] p.56)

> There is no formal high-level security model, the security policy is derived from a collection of low-level detail checking. This is difficult because without a formal model it is impossible to say what *secure* means. With no standard model each vendor is free to reinvent the term secure. In addition no particular implementation can be verified.

New *models* of security that fit well the reality of mobile computing are obviously needed. Even if a security concept makes perfect sense for mobile code, the formal basis of such a concept may have to be entirely redefined. The notion of *noninterference* [9] is a case in point: to require that certain mobile objects can never influence the behavior of other objects is a very natural way of imposing certain security policies. But the automata-theoretic formal basis of noninterference is quite inadequate for a distributed system of communicating objects, so that a new model has to be developed. Even authentication and digital signature mechanisms may have to be reformulated to take account of the adaptable and behavior-changing nature of mobile code.

Our proposal is to adopt rewriting logic as the formal semantic framework on which to build the new security models that will be needed. The point is that, for mobile computing, security properties need to be mathematically formulated in the appropriate semantic framework, one that can express security in an open, object-oriented, highly distributed, and dynamically changing environment. As already mentioned, rewriting logic seems well suited for such a task because of its good formal support for all these features and because it encompasses very naturally the actor models [20, 13]. Furthermore, having rewriting logic as the common formal framework for both security models and for a declarative mobile language like Mobile Maude, whose modules are rewrite theories, will make formal verification of mobile programs much easier than verifying conventional programs, and in particular will greatly ease the verification of security properties.

After more experience has been gained in the definition of adequate models, one should also investigate adequate *security specification languages*, that can make security assertions at an even higher level of abstraction than the executable level provided by rewriting logic and can be rigorously proved correct with respect to rewriting logic specifications. We conjecture that a temporal logic with good primitives for specifying object-oriented properties may provide a good formalism for these purposes. The language in the theories of such a security specification logic should themselves be parameterized by the underlying rewriting theories corresponding to the executable specification level, thus having a tight integration between the logics and the models at both levels. There is independent evidence both form the work of Gray and McLean [10] on specification of cryptographic protocols and from Pfitzmann's work on digital signature schemes [17] indicating that temporal logic is a good specification formalism for these purposes; however, the right choice of primitives and the adequate support of object orientation both deserve further study. Together with those specification formalisms, adequate *proof techniques* supporting verification of security properties should be developed.

The Road Ahead

Although the theoretical foundations of rewriting logic and of its reflective capabilities have been developed in detail [12, 4], and a high performance prototype Maude interpreter already exists [3], much work remains ahead to thoroughly exploit these ideas in mobile computing. Two important research directions that we plan to investigate in detail in the near future are: a detailed design and implementation of Mobile Maude, including support for wrapping conventional code in *foreign interface modules*; and a detailed study of security models for mobile computing based on rewriting logic, including a concurrent object-based version of noninterference and models for cryptographic protocols.

References

[1] G. Agha. Abstracting interaction patterns: A programming paradigm for open distribute systems. In E. Najm and J-B. Stefani, editors, *Formal Methods for Open Object-based Distributed Systems*, pages 135–153. Chapmn & Hall, 1997.

[2] G. Agha, S. Frølund, W. Kim, R. Panwar, A. Patterson, and D. Sturman. Abstraction and Modularity Mechanisms for Concurrent Computing. *IEEE Parallel and Distributed Technology: Systems and Applications*, 1(2):3–14, May 1993.

[3] Manuel G. Clavel, Steven Eker, Patrick Lincoln, and José Meseguer. Principles of Maude. in *Proc.* 1^{st} *Intl. Workshop on Rewriting Logic and its Applications*, ENTCS, North Holland, 1996.

[4] Manuel G. Clavel and José Meseguer. Reflection in rewriting logic. in *Proc.* 1^{st} *Intl. Workshop on Rewriting Logic and its Applications*, ENTCS, North Holland, 1996.

[5] 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.

[6] José Meseguer (ed.). *Proc.* 1^{st} *Intl. Workshop on Rewriting Logic and its Applications*, entcs, north holland, 1996.

[7] S. Frølund. *Coordinated Distributed Objects: An Actor Based Approach to Synchronization.* MIT Press, 1996.

[8] S. Frølund and G. Agha. A language framework for multi-object coordination. In *Proceedings of ECOOP 1993*, volume 707 of *Lecture Notes in Computer Science*. Springer Verlag, 1993.

[9] Joseph Goguen and José Meseguer. Security policies and security models. In *Proceedings of the 1982 Symposium on Security and Privacy*, pages 11–20. IEEE, 1982.

[10] J. W. Gray III and J. McLean. Using temporal logic to specify and verify cryptographic protocols (progress report).

[11] G. McGraw and E. W. Felten. *Java Security.* Wiley Computer Publishing, 1996.

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

[13] 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.

[14] José Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In *Proceedings of the CONCUR'96 Conference, Pisa, August 1996*. Springer LNCS 1119, 1996.

[15] Sun Microsystems. Frequently asked questions—applet security, 1996. Web document at URL http://java.javasoft.com/sfaq/index.html.

[16] Sun Microsystems. Lowlevel security in java, 1996. Web document at URL http://java.sun.com/ by Frank Yellin.

[17] B. Pfitzmann. *Digital Signature Schemes.* Springer-Verlag, 1996. LNCS, Volume 1100.

[18] D. Sturman. *Modular Specification of Interaction Policies in Distributed Computing.* PhD thesis, University of Illinois at Champaign Urbana, 1996.

[19] D. Sturman and G Agha. A protocol description language for customizing failure semantics. In *The 13th Symposium on Reliable Distributed Systems, Dana Point, California*. IEEE, October 1994.

[20] Carolyn L. Talcott. An actor rewriting theory. In *Workshop on Rewriting Logic*, number 4 in Electronic Notes in Theoretical Computer Science, 1996.

[21] N. Venkatasubramanian, G. Agha, and C. L. Talcott. Scalable distributed garbage collection for systems of active objects. In *International Workshop on Memory Management, IWMM92, Saint-Malo*, LNCS, 1992.

[22] N. Venkatasubramanian and C. L. Talcott. A metaarchitecture for distributed resource management. In *Hawaii International Conference on System Sciences, HICSS-26*, January 1993.

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