Distributed mobile systems: specification and automatic verification

G.Ferrari, C.Montangero, L.Semini, S.Semprini
{giangi, monta, semini, semprini}@di.unipi.it

Dip. di Informatica, Univ. di Pisa, Corso Italia 40, 56125 Pisa, Italy

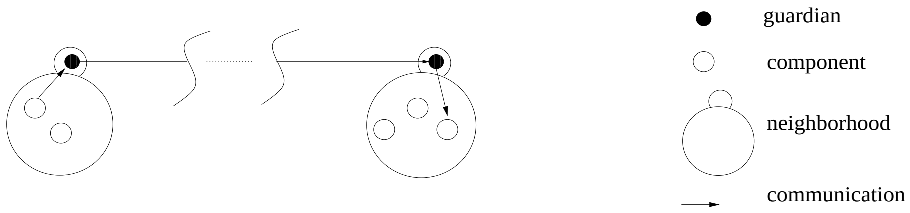
Present–day network computing technologies exploit mobile entities (either *logical*, like program codes and agents, or *physical*, like hand–held devices) that execute certain computational activities while moving around the network. A basic concern in such a context is the complexity of the development of these *network–aware* applications. Network–awareness means that behaviours strongly depend on the network environment of the host in which the application is running. The experience gained to date in the development of network applications has shown the difficulties of using traditional software technologies: reasoning about network applications is subtly different from reasoning about ordinary programs because of stronger requirements on security, different forms of termination, and phenomena like mobility and network-awareness. Therefore, from a Software Engineering perspective there are new challenging issues:

- the definition of structural and computational models to provide designers with conceptual and programming abstractions to master the overall architecture and the structure of the components of network–aware applications;
- the definition of tools to support the specification and verification process thus avoiding "by hand" arguments, which usually leads to problems in the development process.

To support the specification and verification of network applications, we propose to use the network-aware logic Mob_{adtl} and its proof assistant, Mark (Mob_{adtl} Reasoning Kit).

$\mathbf{Mob}_{adtl}$

Mob_{adtl} is a temporal logic based model to specify and develop network–aware applications [1]. In Mob_{adtl}, a network application consists of an immutable net of elaboration nodes: the *neighborhoods*. A neighborhood is a bounded environment where several components (both stationary and mobile) live. Each neighborhood is associated with a stationary component, the *guardian*: the



knowledge of their own guardian makes components location aware. Moreover, guardians monitor the components and limit the resources they can use. The model supports *strong subjective mobility*. Communications are based on *asynchronous message passing*, and communications between components occur via the guardians, which provide also routing facilities to forward messages and to handle migrating components. Component migration requests can be refused by guardians, for instance for security reasons. Indeed, guardians intercept messages and components and decide which messages and components can enter or leave the neighborhood they control.

The notion of neighborhood permits to model several crucial issues of network–aware programming. For instance a neighborhood can represent an administrative domain where components are under the control of a specific authority, a naming domain where components adopt a given naming policy, and also an address space.

Mob_{adtl} comes equipped with a structural model which takes the form of an axiomatic presentation in in Oikos–*adtl* (a linear–time temporal logic specifically designed to deal with distributed systems with asynchronous communications [3]), and a Unity-like inference system.

As shown in the figure, the model constrains messages and migrating components to be routed via the guardians at both ends. Conversely, the model does not embody any routing or security policy, i.e., it does not specify which other guardian may be involved. These policies can be explicitly specified at lower refinement steps, extending the axiomatization with suitable axioms.

We tested Mob_{adtl} with several case studies, including the security requirement of a network application where components have different security constraints and live in different administration domains [2].

Mark

To automatise property verification in Mob_{adtl}, we are working on the definition of the proof assistant Mark (Mob_{adtl} Reasoning Kit). So far, we have defined a prototype implementation for Mark.

The advantages of using this proof assistant include: the ability to handle standard bureaucratic activities (boolean reasoning and standard verification techniques); the ability to support the proving pattern that emerges when composing subsystems; the ability to keep track of the assumptions a property relies on.

Mark consists of several libraries and a back-end theorem prover, Isabelle [4]. In Mark there are three forms of libraries: Mob$_{adtl}$ theories, Mob$_{adtl}$ tactics, and Mob$_{adtl}$ proofs. The theory library contains the translation of Mob$_{adtl}$ axioms into Isabelle theories. The tactic libraries contain proof stereotypes which can be reused in the verification activity. Finally, Mob$_{adtl}$ proof libraries include theorems derived within the assistant.

Mark works as follows. The user supplies it with the system specification. Then, to prove a property, the user proceeds interactively selecting tactics or theorems among those listed by Mark. When the proof terminates, the user can add it to the Mob$_{adtl}$ proof libraries. If a new tactic has been singled out during the proof, it can be elaborated and inserted in the Mob$_{adtl}$ tactic library.

We have validated Mark on several examples. In particular, we defined and verified the properties of a non trivial application where mobile components are allowed to carry some resources (communication channels) with them when moving around the network.

References

[1] G. Ferrari, C. Montangero, L. Semini, and S. Semprini. Mobile Agents Coordination in Mob$_{adtl}$. TR-SAL-4, Technical Report Saladin Project. Also in A. Porto and G.-C. Roman, editors, *Proc. 4th Int. Conf. on Coordination Models and Languages, COORDINATION'00*, volume 1906 of *Lecture Notes in Computer Science*, Limassol, Cyprus, Sep. 2000. Springer-Verlag.

[2] G. Ferrari, C. Montangero, L. Semini, and S. Semprini. Multiple Security Policies in Mob$_{adtl}$. TR-SAL-25, Technical Report Saladin Project. Also in *Proc. Workshop on Issues in the Theory of Security (WITS'00)*, Geneva, 7,8 July 2000.

[3] C. Montangero and L. Semini. Composing Specifications for Coordination. In P. Ciancarini and A. Wolf, editors, *Proc. 3nd Int. Conf. on Coordination Models and Languages*, volume 1594 of *Lecture Notes in Computer Science*, pages 118–133, Amsterdam, April 1999. Springer-Verlag.

[4] L. Paulson and T. Nipkow. Isabelle. www.cl.cam.ac.uk/Research/HVG/Isabelle/.