Mobile Agents Coordination in Mob$_{adtl}$

Gianluigi Ferrari, Carlo Montangero, Laura Semini, and Simone Semprini

Dipartimento di Informatica, Università di Pisa.
{giangi,monta,semini,semprini}@di.unipi.it

Abstract. We present and formalize Mob$_{adtl}$, a model for network–aware applications, extending the Oikos–*adtl* temporal–logic based approach to the specification and verification of distributed systems. The model supports strong subjective mobility of *agents* under the control of stationary *guardians*. Communications are based on asynchronous message passing. The approach exploits the notions of coordination and refinement to deal separately with the specification of functional issues in the agents, and with the specification of coordination policies, e.g. security, routing, etc., in the guardians. The goal is to specify mobile agents as independently as possible of the requirements related to the other facets of distribution. The specification of an application is obtained by instantiating the general model, refining it along different dimensions corresponding to the different aspects of interest, and finally composing the refinements. The main advantage, besides the increased flexibility of the specification process, is that it is possible to specify rich coordination policies incrementally, while the functional units remain relatively simple. We use Mob$_{adtl}$ to specify a simple electronic commerce application, paying particular attention to the incremental specification of the policies. We show how refined policies lead to stronger system properties.

1 Introduction

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. Moreover, the programming focus is on structural or architectural rather than algorithmic issues. The emerging network–aware programming mechanisms and languages [15,10] provide effective infrastructures to support forms of mobility and control of dynamically loaded software components and physical devices.

A certain amount of success has been achieved in the development of network–aware applications over the WEB; however these experiences have shown the difficulties of using traditional software technologies in the context of network–aware computing. Therefore, from a Software Engineering perspective there is a new challenging issue: the definition of structural and computational models to provide designers with conceptual and programming abstractions to master

A. Porto and G.-C. Roman (Eds.): COORDINATION 2000, LNCS 1906, pp. 232–248, 2000.
© Springer-Verlag Berlin Heidelberg 2000

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

the overall architecture and the structure of the components of network–aware applications.

In this paper we present Mob$_{adtl}$, a temporal logic based model to specify and develop network–aware applications. We introduce the model and its axiomatic presentation in Oikos–*adtl* [22], a linear–time temporal logic specifically designed to deal with distributed systems with asynchronous communications. Our approach is based on the notions of *coordination* [7,1] and *refinement calculus* [4]. Coordination provides a powerful conceptual tool to specify and develop systems in a compositional way. The refinement calculus simplifies the design of a system by using incremental development techniques. In Mob$_{adtl}$, we can specify network–aware applications by separating functionality from structural design: coordination provides primitives to glue together independent computational units, like those realizing the functionalities and those realizing security and routing policies. Hence, coordinators are the basic conceptual and programming abstractions to make applications adapt and react to the dynamic changes of their network environments. However, functional specifications cannot abstract completely from the policies: a policy that does not allow a component to enter into a site may have a visible functional effect. Mob$_{adtl}$ provides the necessary hooks to provide a very abstract description of the consequences of the policies, allowing at the same time to postpone to the appropriate points in the refinement process the specific decisions about the policies themselves. Indeed, Mob$_{adtl}$ does not provide directly any specific policy: effective policies must be explicitly specified through suitable refinement steps.

To illustrate how Mob$_{adtl}$ supports system specification we consider electronic commerce. Electronic commerce has many aspects including security, distribution and recovery. It involves strong interaction patterns among the actors (e.g. clients and vendors). These are typical features of network–aware applications having a set of controlled activities with strong interactions over a distributed environment. In the example, we first define the behaviour of a pair of components: a customer, and an agent sent to order a pizza, and derive the overall properties of the application assuming that the involved sites behave as the generic sites of our model. We then refine the application by fixing some policies for these sites, and show which new properties of the application can be derived.

The remainder of this paper is organized as follows. In Section 2, we present our abstract model for network–aware computing. Section 3 reviews Oikos–*adtl*. The axiomatization of the model is subsequently defined in Section 4. In Section 5 we apply the framework to the specification of a simple electronic commerce application. We conclude the paper with some remarks about related works and future research directions.

2 A Model for Network–Aware Computing

This section presents our abstract model for network–aware computing. First, we classify the models for network–aware computing presented in the literature along the following axis.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

The nature of mobile units: They can be any combination of code, data and control [15,10]. Models where only pieces of code can be moved are said to support a *weak* form of mobility, while models where the units of mobility are *processes* (code + control) or *agents* (code + control + data) support *strong* mobility.

There are programming languages designed only to provide the ability of downloading code for execution (e.g. Java [3]). More sophisticated languages support migration of entire computations (e.g. Telescript [29]). A number of *Distributed process calculi* have been proposed as formal models for network–aware programming. We mention the *Distributed Join-Calculus* [14], the *Distributed π-Calculus* [18], the *Ambient Calculus* [6], and the *Seal Calculus* [28]. All these calculi advocate programming models which support strong mobility. Coordination-based models of behaviours have been adopted in the design of the Klaim experimental programming language [11], the $\sigma\pi$ calculus [2] and Mobile Unity [21]. Klaim extends the Linda [16,8] model with multiple distributed tuple spaces and provide programming abstractions to support process mobility. $\sigma\pi$ permits the specification of dynamic networks of components (i.e. networks which reconfigure their communication linkages): a component name is the unit of mobility. In Mobile Unity the unit of mobility is a component. Program states are equipped with a distinguished variable, the *location variable*, and a change of the value of the location variable corresponds to a component migration.

Mobility extent: If not all the components can move, it is useful to distinguish between mobile and *stationary* components. In the Aglets API [19], the *aglet context* provides a bounded environment where mobile components live. Aglet contexts are considered as not transferable. Similarly, Telescript's *places* are stationary components.

The dichotomy between stationary and mobile components also emerges in the foundational calculi. For instance, Klaim's nodes and Distributed π calculus allocated threads are stationary components. In the Ambient calculus, instead, ambients are the units of movement and they can be always moved as a whole including subambients. However, it is difficult to prove behavioural properties of ambients as the control of movements is distributed over all ambients (any of them can exercise the movement capabilities). A type system that constraints mobility of ambients has been proposed in [5]. Using type information one can express whether an ambient is mobile or stationary. To constrain explicitly ambient movements, an extension of the basic calculus has been proposed in [20]: the movement interactions become synchronous operations, and any movement can take place only if the two participant ambients agree.

Location awareness: The units can be either *location aware* or not. Location awareness results in the ability of choosing between a set of possible next actions depending on the current location. Locations reflect the idea of administrative domains and computations at a certain location are under the control of a specific authority.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

Basically, in all models the units are location aware. The notions of ambients in the Ambient calculus, of seals in the Seal calculus, and localities in the Distributed Join calculus and Distributed π calculus correspond to variants of the general notion of locations. Finally, in Mobile Unity, location awareness is modelled by the value hold by the location variable.

Location control: The mobile units can control their location (*proactive* or *subjective* mobility), or can be moved by other entities (*reactive* or *objective* mobility). For instance, in Mobile Unity and Klaim only a proactive form of mobility is allowed, while the *seals* are moved by their parent in the Seal calculus. The Ambient calculus is an hybrid: ambients can decide to move, but they carry their sub–ambients with them, which are thus moved in an objective way.

Communication model: Examples are the transient shared memory of Mobile Unity; the name passing over a named channel in the Distributed Join calculus; the anonymous asynchronous message passing via explicit addressing of Klaim. In general, remote interactions are handled through explicit naming: a component which interacts over a non-local channel has to know the place where the channel is located. An exception to this schema is the Ambient calculus: the knowledge of an ambient name is not enough to access its services; it is necessary to know the route to the ambient. Finally, interposition mechanisms (*wrappers*), which encapsulate components to control and monitor communications have been exploited in [27]. Wrappers support the enforcement of security properties by constraining communications between trusted and untrusted components: wrappers explicitly specify which are the allowed communications among components.

In our model, a system is based on an immutable net of elaboration nodes: the *neighborhoods*. The neighborhood is a bounded environment where several components (both stationary and mobile) live. Components have a unique name, e.g. determined by the initial neighborhood and a suffix to distinguish siblings.

The notion of neighborhood basically corresponds to that of location. Each neighborhood is associated with a stationary component, the *guardian*. The knowledge of their own guardian makes components (both stationary and mobile) location aware. A guardian acts as an interposition interface among components and neighborhoods: it specifies and implements communication and movement policies. In other words, guardians monitor the components and limit the resources they can use. More precisely, communications between components occur via the guardians. Communications are based on *asynchronous message passing*. Guardians provide also routing facilities to forward messages and to handle migrating components.

The model supports strong mobility and mobility is *subjective*, but component migration requests can be refused by guardians, for instance because of security reasons. Indeed, guardians intercept messages and components and can decide which messages and components can enter or leave the neighborhood they control. The following figure provides a pictorial representation of our model.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

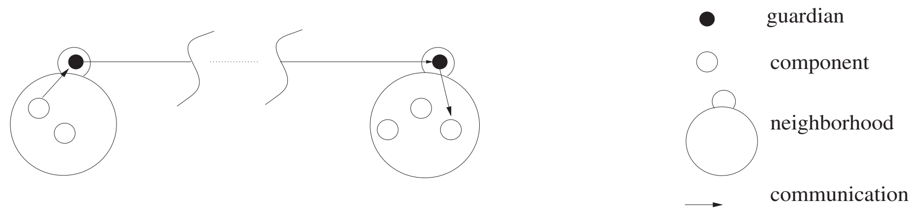

Fig. 1. Communication among components in different neighborhoods: messages and migrating components are routed via the guardians at both ends. Which other guardian may be involved is fixed by the routing and security policies at lower refinement steps.

The notion of neighborhood permits to model several crucial issues of network–aware programming. For instance a neighborhood can be used to represent an administrative domain where components are under the control of a single authority (the guardian), a naming domain where components adopt a given naming policy, and also an address space. The current notion of neighborhood is not complete. For instance, important requirements are not covered: the ability to define new neighborhoods and merge existing neighborhoods is missing.

Asynchronous communication permits to keep the model abstract from any specific communication protocol. The model itself does not embody any routing or security policy for the communications between guardians. Effective routing and security policies must be explicitly specified through suitable refinement steps. The development approach deals separately with functional, security, and mobility issues. For instance, we can fully specify the functionality of a component by giving only very abstract description of the security requirements. A complete system specification is therefore obtained by plugging together different refinements corresponding to different aspects of the system.

3 Background: Oikos–*adtl*

Oikos–*adtl* is a specification language for distributed systems based on asynchronous communications. It is based on an asynchronous, distributed, temporal logic, which extends Unity [9] to deal with components and events.

The language, its semantics, and a number of theorems have been introduced in [22,26]. We recall here the most important concepts.

The Computational Model. A system is characterized by a list of component names, and a set of state transitions, each associated to a component. A computation is a graph of states like the one in the figure below (dotted arrows denote multiple transition steps, plain arrows single transitions or message emission). Any state of component M is either the result of the application of a local transition, (as the one establishing q in the figure below), or the result of a send operation originated in a distinguished component (as message r).

Find authenticated court documents without watermarks at docketalarm.com.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase®
Smarter legal research.