A Mobility Calculus with Local and Dependent Types

Mario Coppo[1,⋆], Federico Cozzi[2,⋆⋆], Mariangiola Dezani-Ciancaglini[1,⋆⋆⋆], Elio Giovannetti[1,†], and Rosario Pugliese[3,‡]

[1] Dip. Informatica, Univ. di Torino, corso Svizzera 185, Torino, Italy
[2] Dip. Scienze Matematiche e Informatiche, Univ. di Siena, pian dei Mantellini 44, Siena, Italy
[3] Dip. Sistemi e Informatica, Univ. di Firenze, viale Morgagni 65, Firenze, Italy

Dedicated to Jan Willem Klop on the occasion of his 60th birthday.

Abstract. We introduce an ambient-based calculus that combines ambient mobility with process mobility, uses group names to collect ambients with homologous features, and exploits co-moves and runtime type checking to implement flexible policies for controlling process activities. Types rely on group names and, to support dynamicity, may *depend* on group variables. Policies can dynamically change also through installation of co-moves. The compliance with ambient policies can be checked *locally* to the ambients and requires no global assumptions. We prove that the type assignment system and the operational semantics of the calculus are 'sound', and define a sound and complete type inference algorithm which, when applied to terms whose type decorations only express the desired policies, computes the minimal type annotations required for their execution. As an application of our calculus, we present a couple of examples and linger on the setting up of policies for controlling the activities of the entities involved.

1 Introduction

The foundational research on distributed and mobile computing, driven by the technological advances of the last decades, has produced a number of theoretical models (for example [27,12,29,7,3], to cite just a few), which can be generally assimilated to some form of distributed process calculus or of 'ambient' calculus.

⋆ Partially supported by EU within the FET - Global Computing initiative, project DART IST-2001-33477.
⋆⋆ Partially supported by EU within the project IHP 'Marie Curie DisCo' HPMT-CT-2001-00290.
⋆⋆⋆ Partially supported by EU within the FET - Global Computing initiative, project MIKADO IST-2001-32222 and MURST Cofin'04 project McTafi.
† Partially supported by EU within the FET - Global Computing initiative, project DART IST-2001-33477.
‡ Partially supported by EU within the FET - Global Computing initiative, project MIKADO IST-2001-32222, and FP6-2004-IST-FET Proactive, project SENSORIA proposal contract number 016004.

A. Middeldorp et al. (Eds.): Processes... (Klop Festschrift), LNCS 3838, pp. 404–444, 2005.

All such models rely on (often sophisticated) type systems to express and check behavioural properties concerning mobility, resource access, security, etc. In most of them, a system or component is represented by a term P of a given calculus, a type $\mathtt{V}$ assigned to P and an environment Σ. In the standard view, as is well-known, the term P abstractly describes the implementation, its type $\mathtt{V}$ may express some behavioural properties, and the environment Σ is a set of assumptions on the outside world. There is thus the notion of a global environment, whose corresponding concrete scenario is one where all the interacting parties are known in advance to each other, so that static checks performed before execution ensure the correctness of the whole system.

In particular, type systems for ambient calculi are usually based on the notion of a process/ambient type which describes the kind of communication and the kind of mobility actions a process can perform and, at the same time, the kind of movements and actions an ambient can make because of the activity of its internal processes. Every ambient name is assigned (by a global assumption or by a name restriction) a type which is simply the type of the processes it is allowed to contain. The basic typing rule for such systems is therefore some variant of the rule:

$$\frac{\Sigma,\ m{:}\mathtt{V} \vdash P : \mathtt{V}}{\Sigma,\ m{:}\mathtt{V} \vdash m[P] \text{ is well typed}}\ (\textsc{Amb})$$

When dealing with computing in wide-area distributed and mobile systems, however, static verification is impractical both because of the huge amount of information to be checked and because typing information could be partial, inaccurate or missing. In such 'open' and dynamic systems no global environment can be assumed; on the contrary, there usually exist several different local and autonomous computational environments. Moreover, interaction may take place between parties whose respective properties are unknown or only partially known to each other. If stopping the execution for re-checking is to be avoided, every potentially dangerous component must dynamically carry with it sufficient behavioural information that can be checked at runtime by the other components interacting with it (see, e.g., the approach based on *proof-carrying code* [33]).

To model these scenarios, we propose here an ambient-based calculus which combines ambient mobility with general process mobility (like $\mathbf{M}^3$ [15]) and where there are no global assumptions on ambient names, since there is no static ambient type. Every ambient name m may be used to build an ambient $m[P]$ with any desired content P; on the other hand, process movements are constrained by the presence of co-moves and by runtime type checking. Indeed, following [22], we define an *operational semantics with types* which exploits types to authorize or block reductions but is simpler than a full-fledged *typed operational semantics*, because it only checks that types agree with process movements.

Types rely on group names, sort of 'family names' that group ambients with homologous features. Mobility properties and co-actions (entrance permissions) are expressed in terms of groups. The notion of a group, however, can be as fine-grained as necessary: in principle, each ambient can be in a distinct group. Dynamicity is enhanced by means of group-dependent types: types can contain