AMBIENT DÉCOR (ABSTRACT)

LUCA CARDELLI AND ANDREW D. GORDON
Microsoft Research
St. George House, 1 Guildhall Street
Cambridge, CB2 3NH, UK
E-mail: luca@luca.demon.ac.uk

Java has demonstrated the utility of type systems for mobile code, and in particular their use and implications for security. Security properties rest on the fact that a well- typed Java program (or the corresponding verified bytecode) cannot cause certain kinds of damage.

In this paper we provide a type system for *mobile computation*; that is, for computation that is continuously active before and after movement. We show that a well-typed mobile computation cannot cause certain kinds of run-time fault. It excludes the communication of values of the wrong kind, anywhere in a mobile system.

In our paper [1] we introduced the (untyped) *ambient calculus*, a process calculus for mobile computation and mobile devices. The untyped ambient calculus is able to express, via encodings, standard computational constructions such as channel-based communication, functions, and agents.

The type system presented here is able to provide typings for those encodings, recovering familiar type systems for processes and functions. In addition, we obtain a type system for mobile agents and other mobile computations. The type system is obtained by decorating the untyped calculus with type information, preserving the untyped reduction semantics.

An ambient is a place where other ambients can enter and exit, and where processes can exchange messages. The first aspect, mobility, is regulated by run-time capabilities and will not be restricted by our type system. The second aspect, communication, is what we concentrate on.

Within an ambient, multiple processes can freely execute input and output actions. Since the messages are undirected, it is easily possible for a process to utter a message that is not appropriate for some receiver. The main idea of our type system is to keep track of the *topic of conversation* that is permitted within a given ambient, so that talkers and listeners can be certain of exchanging appropriate messages.

The range of topics is described by *message types*, W, and *exchange types*, T. The message types are $Amb[T]$, the type of names of ambients that allow exchanges of type T, and $Cap[T]$, the type of capabilities that when used may cause the unleashing of T exchanges (as a consequence of opening ambients that exchange T). The exchange types are Shh, the absence of exchanges, and $W_1 \times \ldots \times W_k$, the exchange of a tuple of messages with elements of the respective message types. For $k = 0$, the empty tuple type is called $\mathbf{1}$; it allows the exchange of empty tuples, that is, it allows pure synchronization. The case $k = 1$ allows any message type to be an exchange type.

One of the original motivations of the ambient calculus was to provide a natural semantics for wide-area network languages. We define a simple agent language inspired by Telescript [4]. We give its semantics in terms of ambients. Moreover, we are able to assign types to our definitions, yielding a typed agent language.

References

1. L. Cardelli and A.D. Gordon. Mobile ambients. In *Foundations of Software Science and Computational Structures,* Maurice Nivat (ed.) LNCS 1378, pages 140–155. Springer-Verlag, 1998.
2. L. Cardelli. Abstraction for mobile computation. *to appear*, 1989.
3. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100(1):1–77, 1992.
4. J.E. White. Mobile agents. In J. Bradshaw, editor, *Software Agents.* AAAI Press / The MIT Press, 1996.