

# A Virtual Machine for the Ambient Calculus

Luis Rodrigo Gallardo Cruz\*, Oscar Olmedo Aguirre†

Sección de computación, Dpto. Ingeniería Eléctrica, CINVESTAV, México, D.F.

\*rodrigo@nul-unu.com, †oolmedo@cs.cinvestav.mx

**Abstract**—Mobile computing is the use of a Wide Area Network as an integral part of a computer system. In spite of similarities with distributed computing research in the area has shown that there are substantial differences between these two disciplines. As a result, theoretical and practical tools from distributed computing can not be used in mobile computing.

There is a great body of work in the area of formal models for mobile computing, including in particular the Ambient Calculus, but little of this work has translated into the development of programming languages for mobile computing, which inhibits experimentation and practical use.

This paper describes the design and implementation of an abstract machine for execution of the Ambient Calculus, which we call MAC. The implementation of the machine includes a tool to help visualize reduction steps in the ambient calculus, for debugging or educational purposes. The contribution of this work is that the availability of this machine will aid in the search for adequate programming languages for mobile computing.

**Keywords**—Mobile computing, Ambient calculus, Abstract machine implementation.

## I. INTRODUCTION

Mobile computing is understood to be the application of computing in the presence of a Wide Area Network, such as the Internet [1]. It differs from distributed computing in that there is no practical way to give reliability, quality of service or scalability guarantees. An application designed for mobile computing must be capable of dealing with service interruptions, long delays and changes of addresses for resources.

In order to fully realize the potential of mobile computing it is necessary to develop models of computation that take these characteristics into account. Over the last few years several such models have been proposed. One of these, the Ambient Calculus, has acquired acceptance in the research community.

The present work builds upon that foundation. We present the design and implementation of an abstract machine that allows the execution of process descriptions in the Ambient Calculus, called MAC. We believe that the availability of MAC will encourage experimentation with the Ambient Calculus as a language for mobile computation, allowing a wider audience of users to better understand its strengths and weaknesses and encourage its possible use in programming mobile devices. It will also allow experimenting with the variations of the calculus that have been proposed in the literature.

## II. RELATED WORK

Over the years several models of concurrent, distributed and mobile computation have been proposed in the literature. The first were CSP [2] and CCS [3] which established communication between processes as the fundamental primitive for concurrent programming. This idea was further explored and

refined in the  $\pi$ -calculus [4], that became the reference for all subsequent developments and is as fundamental for concurrent programming as the lambda calculus is for sequential programming.

Models for distributed computation build upon these bases, but add notions of explicit location of a process. This allows us to model phenomena such as location failure and process migration.

## III. AMBIENT CALCULUS

Mobile computation has special characteristics that make it different from distributed computation. Among these are:

- The network may be divided into individual administrative domains, which do not trust one another.
- There are unavoidable delays in communication. It may not be possible to distinguish between long delay and failure.
- The topology of the network may change at any moment.

In order to include these characteristics into a model of mobile computation, Gordon and Cardelli introduced the Ambient Calculus [5].

The Ambient Calculus takes as its primitive concepts the notion of an Ambient, which is a bounded location where computation takes place. Ambients may be nested within one another and, if they have adequate permissions, move in or out, carrying their complete contents with them. Ambients have both an identity and a name. The knowledge of the name of an ambient allows us to control it. Communication takes place only between processes that live in the same ambient. In order to communicate with a remote process we must create a “traveler” ambient that carries the information back and forth. In terms of expressivity the Ambient Calculus is computationally complete, as it can be used to encode the  $\pi$ -calculus [5].

In this work we have used a slightly modified version of the calculus. In what follows we present a formal description, with a discussion of the changes made.

The calculus has two kinds of primitive entities. First we have expressions, that are values that can be communicated. An expression is either a name, that can be used to create an ambient, a capability, that can be used to control it, or a string of capabilities to be executed in succession. We represent expressions by upper case letters  $M, N$ , etc.

An  $in\ n$  capability instructs the containing ambient to move *into* a sibling ambient of name  $n$ . An  $out\ n$  capability instructs the containing ambient to move out of its parent ambient, if the name of the parent is  $n$ . An  $open\ n$  orders an ambient of



$m[\text{in } n.P \mid Q] \mid n[R] \longrightarrow n[m[P \mid Q] \mid R]$	(In)
$n[m[\text{out } n.P \mid Q] \mid R] \longrightarrow n[R] \mid m[P \mid Q]$	(Out)
$m[\text{open } n.P \mid n[R] \mid S] \mid Q \longrightarrow m[P \mid R] \mid S \mid Q$	(Open)
$m[\langle M \rangle \mid (n).P \mid Q] \longrightarrow m[P\{n := M\} \mid Q]$	(Comm)
$!(x).P \mid \langle M \rangle \longrightarrow !(x).P \mid P\{x \rightarrow M\}$	(BComm)
$!\text{open } n.P \mid n[Q] \longrightarrow !\text{open } n.P \mid P \mid Q$	(BOpen)
$n[!\text{in } m.P \mid Q] \mid m[R] \longrightarrow m[n[!\text{in } m.P \mid P \mid Q] \mid R]$	(BIn)
$n[m[!\text{out } n.P \mid Q] \mid R] \longrightarrow n[R] \mid m[!\text{out } n.P \mid P \mid Q]$	(BOut)
$P \longrightarrow Q \implies n[P] \longrightarrow n[Q]$	(RedAmb)
$P \longrightarrow Q \implies (\nu n)P \longrightarrow (\nu n)Q$	(RedRes)
$P \longrightarrow Q \implies P \mid R \longrightarrow Q \mid R$	(RedPar)

Fig. 2. Operational rules for the ambient calculus

$$\begin{aligned}
& \text{Home}[\text{open } n \mid \text{Agent}[\text{out Home.in Home.n}[\text{out Agent.open Agent.P}]]] \\
& \longrightarrow \text{Agent}[\text{in Home.n}[\text{out Agent.open Agent.P}]] \mid \text{Home}[\text{open } n] \\
& \longrightarrow \text{Home}[\text{open } n \mid \text{Agent}[n[\text{out Agent.open Agent.P}]]] \\
& \longrightarrow \text{Home}[\text{open } n \mid n[\text{open Agent.P}] \mid \text{Agent}[]] \\
& \longrightarrow \text{Home}[\text{open Agent.P} \mid \text{Agent}[]] \\
& \longrightarrow \text{Home}[P]
\end{aligned}$$

Fig. 3. An example trace

composition must be simulated by interleaving execution of the components.

The rule *In* (figure 5) is a semantic rule. It describes the execution of an *in* *n* capability and is applied when the required sibling ambient exists. Although complex looking, most of it is basically bookkeeping. It searches the wait queues of ambients around it for processes that should be moved back into the run queue (for example, a process inside the moving ambient that is waiting to execute an *out* *n*).

The other MAC rules have a similarly complex structure and for this reason are not presented here.

MAC was proved correct by giving a translation from machine states to ambient calculus expressions. This gives a translation between machine rules and ambient calculus transformations. It was then shown that these transformations are legal ambient calculus transitions. For the internal rules this is true because they translate to, at most, structural transformations, such as rearrangement of the order of parallel compositions. For the semantic rules it is only slightly more involved. The book keeping translates to structural modifications, while the semantic content translates directly to one of the operational rules of the calculus.

A complimentary result of completeness is not possible, since the ambient calculus, as most other process calculus, is fundamentally non-deterministic. MAC, on the other hand, is deterministic, and thus it is only able to obtain one of the possibly many final results of a process' execution.

#### A. Implementation

The implementation of MAC was made in the Haskell programming language, a functional, lazily evaluated, strongly

typed language [8]. The characteristics of the language made it possible for the implementation to be almost entirely a transliteration of the machine rules to Haskell syntax. In places where the design is not completely specified arbitrary choices were made. This implementation is available for download at <http://www.nul-unu.com/quien/rodrigo/MAC>.

The implementation includes an utility that prints machine states as a series of nested boxes, representing each an ambient. This allows easier visualization of a process' trace of execution, for either debugging or explanatory purposes.

#### B. Case of study

We present as an example of the Ambient Calculus, an implementation of a RPC protocol.

$$(\nu n)(io[M.(\langle a \rangle \mid \text{open } res.(x).n[M^{-1}.\langle x \rangle]]) \mid \text{open } n) \mid (x).P$$

The process, as printed by MAC after a few internal steps, is shown in figure 6.

In this example the ambient *io* travels to a remote site (following the path *M*) which contains a process *open io* that allows the argument  $\langle a \rangle$  to be output. Once computation ends, the remote ambient produces a *res* ambient that is captured by the waiting *n* ambient and sent back to the original location.

## V. CONCLUSION AND FUTURE WORK

We have successfully designed and implemented a correct abstract machine for execution of the ambient calculus. This machine will facilitate teaching and experimentation with the calculus. The design uncovered a minor inconvenience in the original calculus, which was solved by the introduction of guarded replication.

