Specifying Secure Mobile Applications

Andrew Phillips\textsuperscript{1}, Susan Eisenbach\textsuperscript{2}, Bashar Nuseibeh\textsuperscript{3}, and Nobuko Yoshida\textsuperscript{2}

\textsuperscript{1} Microsoft Research, 7 JJ Thomson Avenue, Cambridge CB3 0FB
\texttt{andrew.phillips@microsoft.com}
\textsuperscript{2} Imperial College London, 180 Queen’s Gate, London SW7 2AZ
\texttt{\{sue,yoshida\}@imperial.ac.uk}
\textsuperscript{3} The Open University, Milton Keynes, MK7 6AA
\texttt{b.nuseibeh@open.ac.uk}

Abstract. Ambient calculi are a promising formalism for specifying mobile computation, which benefit from a range of analysis techniques. However, Ambient calculi have been designed mostly as minimal models for mobility, rather than as specification languages for mobile applications. This paper describes a variant of Ambients, the Channel Ambient calculus, which is designed to be at a level of abstraction suitable for specifying secure mobile applications on distributed TCP/IP networks. A graphical representation for the calculus is also presented, allowing specifications to be written in either textual or graphical format, together with a number of mechanisms that can be used to enhance the security of mobile applications. The calculus has the advantage that it can be directly mapped to program code, which is executed by a runtime system based on a provably correct abstract machine.

1 Introduction

Mobility is an important paradigm for modern distributed applications, yet mobile applications are difficult to develop. Not only do they involve complex parallel interactions between multiple components, but they must also satisfy strict security requirements. It has been argued [1,2,3] that the development of such applications would benefit from a specification language based on a formal model. Foundational research [2] has shown that Ambient calculi are a promising approach to specifying mobile computation, which benefit from a range of analysis techniques. Such calculi can be classified as operational specification languages in the sense of [4], which are typically well-suited for rapid prototyping and development of applications. However, Ambient calculi have been designed mostly as minimal models for mobility, rather than as specification languages for mobile applications.

To address this issue, the Channel Ambient calculus [5] was recently developed to be at a level of abstraction suitable for specifying mobile applications. An abstract machine for the calculus was also defined, and was proved both sound and complete with respect to the calculus. Based on this research, a programming language and runtime system for the calculus have also been implemented.
[6], together with an application for tracking the location of migrating ambients [5]. The correctness of the abstract machine ensures that the work done in specifying and analysing mobile applications is not lost during their implementation. Following on from this work, this paper outlines the design rationale for the Channel Ambient calculus, and shows how it is both abstract enough to model the essential features of mobile applications, and detailed enough to be readily implemented in a distributed setting. A corresponding graphical representation for the calculus is also presented, together with a number of mechanisms that can be used to reason about the security of mobile applications.

The paper is structured as follows. Section 2 describes the design principles of the calculus and Section 3 presents its formal definition, together with a corresponding graphical representation. Section 4 describes how the calculus is used to specify mobile applications in a distributed setting, and Section 5 describes how the calculus is used to reason about the security properties of mobile applications. Finally, Section 6 uses the calculus to specify an example mobile application, in which a mobile agent monitors resources on a remote site.

2 Design Principles

In order to design the Channel Ambient calculus, a number of assumptions were made about the components of mobile applications and the way in which these components interact, as outlined below. This combination of assumptions distinguishes the Channel Ambient calculus from related calculi, as discussed in Section 7.

Mobile applications consist of hierarchical named components, which can be sites, agents or modules. The topology of sites on the Internet is generally hierarchical. For example, Local Area Networks are usually logically contained inside a gateway, which connects them to a Wide Area Network. The gateway can also act as a security barrier or firewall, which protects the sites inside the gateway from outside attacks, and regulates access to the outside network. Agents in a mobile application can also form hierarchies. A given agent is logically contained inside a site, and the agent itself can contain sub-agents and modules. When an agent moves, the sub-agents and modules move with it, and the internal hierarchy of the agent is preserved. Modules can also be hierarchical. A given module can logically contain its own private methods, data and sub-modules, which are not directly accessible from outside the module.

Only adjacent components can interact directly. This is a natural consequence of the hierarchical nature of components in mobile applications. A site inside a LAN cannot interact directly with a site outside the LAN, but needs to do so via a gateway. Similarly, a mobile agent cannot directly interact with the components inside another agent, but needs to do so via the containing agent. Likewise, a module cannot directly access the private data and methods of another module, but needs to do so via the module interface. The notion of adjacent components is illustrated in Fig. 1a. According to the figure, a given component in a hierarchy is adjacent to its parent, its siblings and its children. For example, the
component $a$ is adjacent to components $b, c, d, e$ and $f$, where component $c$ is the parent of $a$, components $b$ and $d$ are its siblings and components $e$ and $f$ are its children. This notion of adjacency places a number of constraints on the way in which components can interact with each other. At best, a component will be able to move into a sibling, move out of its parent, or communicate with its siblings, parent or children. This is illustrated in Fig. 1b, where communication and migration are represented by solid and dotted arrows, respectively.

*Communication and migration are two distinct forms of interaction.* From a security perspective, receiving a message involves potentially less risk than receiving an agent containing program code. Therefore, at a high level of abstraction there should be a logical distinction between these two forms of interaction. Even at the implementation level it can be useful to distinguish between communication and migration, since sending a message involves significantly less overhead than sending a migrating agent, which needs to be stopped, moved through the network and re-started, and in many cases will need to be scanned on arrival for potential malicious behaviour.

*Components communicate over channels.* In some of the most widely-used Internet protocols, such as TCP/IP and UDP, components in the network communicate with each other using channels or *ports*. In general a given server, identified by its IP address, can provide multiple concurrent services on different ports. Similarly, a given module often provides multiple concurrent services, where the name of each service corresponds to a separate channel.

*Peer components are able to communicate directly.* Most peer-to-peer applications rely on direct communication between peers, including many well-known content-distribution applications. Modules also allow direct communication between peers, where modules at the same level in a hierarchy can communicate with each other directly.

*The receiver of a message does not generally know the identity of the sender.* When a TCP/IP connection is established, the receiver does not specify the IP address of the sender, but only specifies the port number. This communication
model allows a given site to provide public services for which the identity of the sender is not known beforehand. Similarly, when a module is developed the identity of potential client modules is not generally known beforehand. Components move in and out of each other over channels. As with communication, channels provide a flexible means of regulating agent migration. A channel is like a key: many agents can use the same key in order to gain access to a site, or each agent can have its own private key. Not only does an agent need a key to enter a site, but it also needs a key to leave the site. This can be used to preserve the confidentiality of data that should only be accessible from within a particular site. In general, many computer systems use personal firewalls to prevent unauthorised programs or agents from accessing the Internet. This prevents such programs from consuming bandwidth without the consent of the user, and also prevents spyware from sending out covert information about the current state of the user’s site. At an implementation level, the TCP/IP protocol also requires two sites to interact over a common channel. If agent migration is to be implemented based on such a protocol, migration will also need to take place over channels.

3 Specification Language

The Channel Ambient calculus uses the notion of an ambient, first presented in [2], to model the components of a mobile application. An ambient is an abstract entity that can be used to model a site, a mobile agent or a module. In keeping with the design principles outlined in the previous section, ambients are named, arranged in a hierarchy and can interact by sending messages to each other and moving in or out of each other over channels. There is also a noticeable symmetry between the actions for communication and migration.

The syntax of the calculus is described in terms of processes $P, Q, R$, actions $\alpha$ and values $a, b, \ldots, z$, where $a, b, c$ represent ambient names, $x, y, z$ represent channel names, $n, m$ represent arbitrary names and $u, v$ represent arbitrary values:

- **0 Null** does nothing and represents the end of a process.
- $P | Q$ **Parallel Composition** executes process $P$ in parallel with process $Q$.
- $\nu n P$ **Restriction** executes process $P$ with a private name $n$.
- $a \cdot P$ **Ambient** executes process $P$ inside ambient $a$.
- $! \alpha . P$ **Action** tries to perform action $\alpha$ and then execute process $P$.
- $\alpha . P$ **Replication** repeatedly tries to perform action $\alpha$ and then execute $P$.

Each action $\alpha$ can involve either a communication or a migration:

- $a \cdot x(v)$ **Sibling Output** sends value $v$ on channel $x$ to a sibling ambient $a$.
- $x^\top(v)$ **Parent Output** sends value $v$ on channel $x$ to the parent ambient.
- $x^\top(u)$ **External Input** receives value $u$ on channel $x$ from a sibling ambient.
- $x(u)$ **Internal Input** receives value $u$ on channel $x$ from a child ambient.
- $\text{in} \; a \cdot x$ **Enter** enters a sibling ambient $a$ over channel $x$. 

4
Definition 1. Graphical Syntax of CA

\[
\begin{array}{c}
\text{Definition 1. Graphical Syntax of CA}
\end{array}
\]

Definition 2. Graphical Reduction in CA

\[
\text{out } x \text{ Leave leaves the parent ambient over channel } x.
\]

\[
\text{in } x \text{ Accept accepts a sibling ambient over channel } x.
\]

\[
\text{out } x \text{ Release releases a child ambient over channel } x.
\]

A corresponding graphical syntax is described in Definition 1, which is reminiscent of various graphical notations such as [7]. Each parallel process is represented as a vertical bar and each component is represented as a box above a process, labelled with the component name. Unlike many notations, each vertical bar is also labelled with the current state of the process, and a process with a
private name is represented as a dotted ring around the process, labelled with the name. Standard syntax conventions are used, including assigning the lowest precedence to the parallel composition operator and writing $\alpha$ as syntactic sugar for $\alpha.0$. In addition, Local Output $x(n)$ and Child Output $a/x(n)$ are defined as syntactic sugar using parent output and sibling output inside a private child ambient, respectively, as described in Appendix A.

The execution rules of the calculus are described in Definition 2. Each rule describes how a given process $P$ can evolve to $P'$ by performing a single execution step, where $P_{(v/u)}$ assigns the value $v$ to $u$ in process $P$:

1. If an ambient $a$ contains a sibling output $b \cdot x\langle v \rangle.P$, and there is a parallel sibling ambient $b$ with an external input $x^\uparrow(u).Q$, then the value $v$ can be sent to ambient $b$ along channel $x$, and assigned to the value $u$ in process $Q$.

2. If an ambient $a$ contains a parent output $x^\uparrow\langle v \rangle.P$, and there is an internal input $x(u).Q$ in parallel with $a$, then the value $v$ can be sent along channel $x$, and assigned to the value $u$ in process $Q$.

3. If an ambient $a$ contains an enter $\mathbf{in} b \cdot x.P$, and there is a parallel sibling ambient $b$ with an accept $\mathbf{in} x.Q$, then $a$ can enter $b$ over channel $x$.

4. If an ambient $a$ contains a leave $\mathbf{out} x.P$, and there is a parent ambient with a release $\mathbf{out} x.Q$, then $a$ can leave its parent over channel $x$.

A reduction can also occur inside a parallel composition (5), inside a restriction (6) or inside an ambient (7), and equal processes can perform the same reduction (8), where the $P \equiv Q$ states that the process $P$ is equal to the process $Q$. The full definition of equality between processes is given in Appendix A. Processes are equal up to re-ordering of parallel compositions and bound names, and a replicated action $\alpha.P$ is equal to $\alpha.(P \mid \alpha.P)$. This allows the replicated action to repeatedly perform the action $\alpha$ followed by the process $P$.

4 Distributed Model

The Channel Ambient calculus can be used to model mobile applications that execute in a wide range of networks. In this paper, mobile applications are assumed to execute on networks that support the widely used TCP/IP version 4 protocol. In order to model the properties of such networks, the syntax of the calculus is constrained to distinguish between two types of ambients: sites $s$ and agents $g$. Sites represent hardware devices that have a fixed network address, while agents represent software programs that can move in and out of sites and other agents:

Sites A process inside a site $s$ is constrained so that it cannot contain a sibling output to an agent. This reflects the assumption that agents do not have a network address, and therefore cannot be reached directly by sites over a network. In addition, a process inside a site cannot contain an enter or a leave. This reflects the assumption that sites have a fixed network address. Note that this constraint does not prevent a site from physically moving around in the network.
Agents A process inside an agent $g$ is constrained so that it cannot contain a site. This reflects the assumption that hardware sites cannot be contained inside software agents.

In practice, the name of a site corresponds to an IP address, while the name of an agent corresponds to a simple identifier. In a hierarchical topology, sites can be logically contained inside other sites to form Local Area Networks. As a result, a given site is unable to communicate directly with another site that is not in the same LAN. For such hierarchical networks it is useful to distinguish between two types of sites: ordinary sites and gateways, which can logically contain other sites to form Local Area Networks. In practice, a gateway acts as a bridge between two networks: the local network it contains and the global network in which it is contained. As a result, a given gateway is usually assigned two network addresses, one for the local network and one for the global network. At the calculus level, the name of a gateway corresponds to its global address. The local address is not needed, since child ambients do not explicitly use the name of their parent in order to interact with it. Therefore, the local address is used merely as an implementation mechanism, to allow messages or agents from child ambients to be correctly routed to their parent gateway.

An example of how sites can be used to model the topology of Local and Wide Area Networks is illustrated in Fig. 2, where each site executes on a separate machine with a given IP address. The sites 192.168.0.2 - 192.168.0.4 are part of a Local Area Network inside a gateway with local address 192.168.0.1 and global address 82.35.60.43. The ambients inside the LAN will send messages or agents to a default parent, which will automatically be routed to the local address of the gateway. The ambients outside the LAN will send messages or agents to the global address of the gateway. Thus, the local and global addresses allow the gateway to distinguish between local and global interactions.
The Channel Ambient calculus can also be used to model the execution of mobile applications on networks that support the TCP/IP protocol. An abstract model of the protocol can be defined by making the following simplifying assumptions:

- During a normal session of the protocol, a given host with address IP$_1$ sends data to a host with address IP$_2$ on a port number $n$.
- If any errors occur during the transmission then the session is aborted.

A host with address IP$_1$ can be modelled as a site with name IP$_1$, and a port number $n$ can be modelled as a channel with name $n$. Communication between hosts can be modelled using the communication primitives of the calculus. For example, a server with IP address 82.35.60.43 running an ftp service on port 21 and a telnet service on port 23 can be modelled as a site with name 82.35.60.43 containing replicated external inputs on channels 21 and 23. A corresponding client with IP address 82.35.60.43 that interacts with the server can be modelled as a site with name 82.35.60.43 containing a sibling output to the server on the corresponding channels:

The migration of agents between hosts on the network can also be modelled using the primitives of the calculus. For example, a server with IP address 82.35.60.43 that accepts an agent on port 3001 can be modelled as a site with name 82.35.60.43 containing an accept on channel 3001.

A private communication channel established between two hosts can be modelled using channel restriction. For example, a private ssh channel established between a client and a server using the SSH protocol is modelled as:

The use of restriction to limit the scope of the ssh channel between client and server guarantees, at an abstract level, that other entities in the network cannot interfere with communication on this channel. For a more detailed model, encryption and decryption mechanisms can be added as a simple extension to the calculus, in the style of [8,9].

Table 1 describes how the reduction rules of the Channel Ambient calculus can be used to model the execution of mobile applications on TCP/IP networks. When two ambients interact over a network the channel corresponds to a port number, and when two ambients interact locally the channel corresponds to a simple identifier. Up to this point, TCP/IP networks have been modelled in the calculus by mapping IP addresses to sites and port numbers to channels. Since a given device is usually assigned a single IP address per network, this approach assumes that each site on the network corresponds to a separate device. A more
Table 1. Using the reduction rules of CA in a distributed setting.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Application in a distributed setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Ambient $a$ sends value $v$ to site $b$ on port $x$</td>
</tr>
<tr>
<td>(2)</td>
<td>Agent $a$ sends value $v$ to parent site on channel $x$</td>
</tr>
<tr>
<td>(3)</td>
<td>Site $a$ sends value $v$ to parent site on port $x$</td>
</tr>
<tr>
<td>(4)</td>
<td>Agent $a$ enters site $b$ on port $x$</td>
</tr>
<tr>
<td>(5)</td>
<td>Agent $a$ leaves site $b$ on channel $x$</td>
</tr>
<tr>
<td>(6)</td>
<td>A process $P$ can reduce inside a restriction</td>
</tr>
<tr>
<td>(7)</td>
<td>A process $P$ can execute independently of other processes in the network</td>
</tr>
<tr>
<td>(8)</td>
<td>The contents of a site $a$ can execute independently of other sites in the network</td>
</tr>
<tr>
<td>(9)</td>
<td>Processes across a network can execute in any order.</td>
</tr>
</tbody>
</table>

flexible approach is to map each site to a socket address consisting of an IP address and a port number. This allows a given device to contain multiple sites, where the number of sites is limited only by the number of available ports. Arbitrary channel names can then be used to interact with a given site, resulting in a more flexible and dynamic interaction model. This can be implemented by adding a thin layer of multiplexing above the TCP/IP protocol, in order to allow a given site to interact on an arbitrary number of named channels.

5 Security Properties

The Channel Ambient calculus can be used to reason about the security properties of mobile applications in a variety of ways. The calculus was designed with a number of security mechanisms built-in, such as the ability to use private channels for both communication and migration. Communication on a private channel prevents messages from being intercepted, and migration on a private channel prevents components from entering or leaving without permission. In this sense, private channels function as keys or passwords, which can be given to a single component or group of components. The formal definition of the calculus also enables a number of safety properties to be proved about the calculus execution model. In particular, it has been shown that an application specified in the calculus cannot produce any runtime errors during execution [10]. The remainder of this chapter discusses two extensions to the calculus that can also help to enhance application security.

Interaction in the Channel Ambient calculus takes place over channels. Therefore, various type systems for channel interaction in the $\pi$-calculus can also be applied to Channel Ambients. One example is the type system presented in [11], which ensures that values are always sent and received on channels of the correct type. This type system can be applied to the Channel Ambient calculus by defining additional rules for the creation of ambients and for the migration of ambients over channels. The syntax of types in the Channel Ambient calculus is described in [10], where each name $n$ is assigned a type $\tau$, which can
be an ambient amb, a channel $\langle \tau \rangle$ carrying values of a given type, a migration channel $\langle \text{mv} \rangle$ or the type of a base value. Although the typing rules are relatively straightforward, they provide a convenient and powerful mechanism for preventing errors during program execution. This is achieved by ensuring that well-typed processes remain well-typed after each execution step [10].

When specifying mobile applications, the ability to create multiple ambients with the same name can sometimes be seen as a security risk. For example, a message being sent to an ambient $a$ over a channel $x$ can be intercepted by an imposter with the same name listening on the same channel:

$$b \cdot x(\text{v}).Q \quad a \quad x[u].R \quad \text{Imposter} \quad z^!(u).P \quad A$$

Similarly, an ambient attempting to enter an ambient $a$ over a channel $x$ can become trapped by an imposter with the same name accepting ambients on the same channel:

$$b \cdot \text{in}_a x.Q \quad a \quad \text{Imposter} \quad x^! (u).R \quad A$$

In the Channel Ambient calculus, these issues can be resolved by using private channels for entering an ambient or sending a message to an ambient. However, it can be cumbersome to require every communication or migration to take place over a private channel, particularly since the privacy of these channels may change dynamically over time. Instead, it is more convenient for an ambient to freely distribute its name and use publicly available channels for communication and migration, without the risk of impersonation. This is achieved by placing a simple constraint on the syntax of the Channel Ambient calculus, which prevents names that are received over channels from being used to create new ambients. A formal definition of the constrained calculus together with a corresponding proof of the prevention of impersonation is presented in [10].

6 A Resource Monitoring Application

Mobile agents can be used to monitor resources on a remote server in order to enhance application performance. For example, the server can be a stock market trading site, the resource can be a list of stock prices published by the server and the client can be a trader, who wishes to respond to changes in stock prices by buying or selling shares accordingly. The client can program an agent with sophisticated trading algorithms, and then send this agent to the server. The agent will be able to respond immediately to changes in stock prices on the server itself, without being affected by network congestion. In addition, the client machine no longer needs to remain connected to the server. This is particularly relevant for lightweight clients with intermittent connection to the network, such as Internet-enabled phones and handheld devices. Furthermore,
if the agent license expires while the client is disconnected, the agent can move to a renewal site, negotiate the renewal of the license and then return to the server to continue trading. Another example of resource monitoring comes from the world of electronic commerce, where the server can be an Internet auction site, the resource can be an item for sale and the client can be a registered user of the site. As with the previous example, the client can program an agent with sophisticated bartering algorithms and perhaps a maximum purchase price. The agent could also be programmed to perform a range of tasks on completion of the purchase. For example, if the purchase is an airline ticket, the agent could then proceed to book accommodation and organise local tours. In this sense, a mobile agent can act very much like a Personal Assistant. The client can give a list of instructions to the agent, and then disconnect or focus on other tasks. From time to time, the client can monitor the performance of the agent or authorise payments above a certain threshold, but the bulk of the work is done by the agent itself.

An execution scenario for the resource monitoring application is described in Fig. 3. The scenario uses a lightweight version of the graphical calculus, in which the states of processes are not shown and the interactions between processes are annotated with channel names and values:

1. The client tries to register with the server by sending its name and a private acknowledgement channel \textit{ack} to the server on the \textit{register} channel.
2. The server completes the registration by sending a private \textit{login} channel to the client on the acknowledgement channel.
3. The monitor agent leaves the client on a private \textit{logout} channel.
4. The monitor agent enters the server on the login channel, and executes the process \textit{P}, which monitors the resource on the server. In parallel, the server executes the process \textit{R}, which forwards information about the resource to the monitor.

Based on this scenario, a calculus specification of the resource monitoring application can be derived in a straightforward manner, as shown in Fig. 4. This is achieved by specifying a corresponding action and co-action for each of the interactions described in Fig. 3. Initially, the client tries to send its name and a private acknowledgement channel \textit{ack} to the server on the \textit{register} channel, and then tries to receive a login channel \textit{x} on the acknowledgement channel. The monitor agent then tries to leave on a private \textit{logout} channel, enter the server on the login channel and execute the process \textit{P}. In parallel, the client tries to release an agent on a private \textit{logout} channel and then execute the process \textit{Q}. The server continually listens on the \textit{register} channel for a client name \textit{c} and an acknowledgement channel \textit{k}. Each time a registration request is received, the server tries to send a private \textit{login} channel to the client on the acknowledgement channel, accept an agent on the login channel and then execute the process \textit{R}. The client, server and network can also execute other processes in parallel, represented by \textit{C}, \textit{S} and \textit{N}, respectively. These processes can potentially try to interfere with the interactions between the client and the server. According to the calculus ex-
execution rules in Definition 2, the execution steps 1-4 of the calculus specification in Fig. 4 are compatible with the scenario described in Fig. 3.
A number of security properties can be ascertained from the calculus specification. Initially, the client sends a private acknowledgement channel to the server. This prevents other processes inside the client from eavesdropping on the acknowledgement channel and stealing the login. On receiving a registration request, the server sends a private login channel to the client over the acknowledgement channel. The login channel acts as a key, which the client can use to send a monitor agent to the server. The server will only allow a single monitor to enter using this key, thereby ensuring strict access control to the server. Similarly, the monitor agent can only leave the client on a private logout channel. This prevents other agents that do not know the name of the logout channel from leaving the client without permission. The properties of the calculus ensure that the application will not produce any runtime errors during execution, and simple constraints on the syntax of the calculus ensure that the monitor agent cannot be impersonated. In addition, the \textit{ack} and \textit{register} channels can be typed as \langle\langle mv\rangle\rangle and \langle amb, \langle\langle mv\rangle\rangle\rangle, respectively, to ensure that they can only be used to receive a login channel and a client name with an acknowledgement channel, respectively.

\section{Related Work}

The Channel Ambient calculus (CA) is inspired by previous work on calculi for mobility, including the Ambient calculus [2], the Nomadic \(\pi\)-calculus [3] and variants of the Boxed Ambient calculus [12]. In many respects, the calculus can be viewed as a variant of Boxed Ambients in which channels are defined as first class entities. The main differences with Boxed Ambients are that ambients in CA can interact using named channels and that sibling ambients can communicate directly. Sibling communication over channels is inspired by the Nomadic \(\pi\)-calculus, although sibling seals cannot communicate directly. The use of channels for mobility is inspired by the mechanism of passwords, first introduced in [14] and subsequently adopted in [15]. The main advantage of CA over existing variants of Boxed Ambients is its ability to directly express high-level constructs such as channel-based interaction and sibling communication. The main advantage of CA over the Nomadic \(\pi\)-calculus is its ability to regulate access to an ambient by means of named channels, and its ability to model computation within nested locations, both of which are lacking in Nomadic \(\pi\).

The constraints on the syntax of the Channel Ambient calculus for preserving ambient authenticity can be viewed as a compromise between the capability model of the Ambient calculus and the unique identifiers of the Nomadic \(\pi\)-calculus. In the Ambient calculus, capabilities are used to enter, leave or open an ambient without revealing the ambient’s name. Capabilities are not as flexible as channels, since they cannot be revoked and there is no limit on the number of times they can be used. Furthermore, when an ambient receives a capability it has no way of knowing the identity of the ambient to which the capability can be applied. At the other extreme, the Nomadic \(\pi\)-calculus requires all agents to
be created with unique, private identifiers. This ensures that no two ambients can have the same name, thereby preventing ambient impersonation. However, in some cases it can be desirable for multiple ambients to have the same name. For example, multiple sites may wish to contain the same service ambient with the same public name, so that agents can invoke the same services on different sites via a uniform interface. The constrained Channel Ambient calculus aims to combine the best of both worlds, by allowing multiple ambients to be created with the same public name, while still allowing ambients with private names to protect their identity.

In the Safe Ambient calculus [16], co-actions are used to allow an ambient to enter, leave or open another ambient. However, there is no way of controlling which ambients are allowed to perform the corresponding actions. For example, a host ambient wishing to accept two guests can give an entry capability to each guest. However, one of the guests could send its entry capability to a friend, who could then use this capability to take the place of the second guest. This situation is possible since the host has no way of distinguishing between guests. In contrast, the Channel Ambient calculus uses channels to regulate access to an ambient. This allows the host to create two private channels, one for each guest, and to only accept one guest on each channel. This ensures that no ambient can take the place of a guest without their consent.

One of the main characteristics of the Channel Ambient calculus is that it allows sibling ambients to communicate directly. This is often desirable within a single machine, or between two machines in a local area network. Even over a wide area network, certain protocols such as TCP/IP provide a useful abstraction for synchronous communication. In cases where asynchronous communication is required, such as the UDP protocol, communication between machines can be made asynchronous by requiring messages to be sent via an intermediate router ambient. The Nomadic $\pi$-calculus highlights the benefits of allowing synchronous communication between agents on the same site. In contrast, Boxed Ambient calculi typically require all communication between sibling agents to take place via the parent. However, there is nothing to prevent one or both of these agents from migrating while the message is still in transit, leaving the undelivered message stuck inside the parent. In order to avoid such forms of message loss, locking mechanisms need to be put in place to ensure that any undelivered messages are retrieved by an agent before it migrates. Such encodings are non-trivial, and place strict constraints on when an agent can and cannot move.

More generally, there are numerous alternative specification languages for mobile systems that are complementary to the process calculus approach. These include MobiS [17], which is based on a tuple-space model and specifies coordination by multi-set rewriting, the IOA Language [18], which is based on the I/O automaton model for reactive systems, and Mobile UNITY [19], which is based on automata that combine via shared variables instead of shared actions. A detailed comparison between process calculi, I/O automata and Mobile Unity is described in [3], which argues that process calculi are perhaps the closest to a programming language for mobility. This makes them an ideal candidate for
bridging the gap between the specification and implementation of mobile applications.

8 Conclusion

This paper presented a calculus for specifying mobile applications, known as the Channel Ambient calculus. The calculus was inspired by previous work on calculi for mobility, and its design was directly influenced by the properties of modern mobile applications. A graphical representation for the calculus was also presented, allowing specifications to be written in either textual or graphical format. The calculus was developed for specifying mobile applications on networks that support the TCP/IP protocol, and the use of channels in the calculus is directly compatible with the use of ports in TCP/IP. Various security mechanisms developed for related calculi were applied to the Channel Ambient calculus, in order to reason about the security properties of mobile applications. These include safety properties to ensure absence of runtime errors, type systems to ensure reliable channel communication, and syntax constraints to ensure agent authenticity. By remaining in the Ambient paradigm, the calculus also benefits from a range of security mechanisms developed for Ambient calculi, most notably Ambient Logics.

In addition to the principles outlined in Section 2, the design of the Channel Ambient calculus was strongly influenced by practical experience in developing a programming language and runtime system for mobile applications [10,6]. A number of extensions to the calculus can also be envisaged, including notions of time and choice. A promising approach to modelling time in process calculi is presented in [20], and a model of non-deterministic choice is defined in [21], both of which could readily be incorporated into the Channel Ambient calculus. Although the calculus has been designed for networks that support the TCP/IP protocol, it could in principle be applied to a range of networks types, including mobile ad-hoc networks. Indeed, mobile wireless networks were some of the first to be modelled by process calculi [22]. Furthermore, although the calculus was developed specifically for modelling mobile applications, it can also be used to model conventional distributed applications, while giving programmers the flexibility to extend these applications with mobile technology as appropriate.

In the long term, a variant of the graphical representation described in this paper could perhaps be used to help elaborate the specification of mobile applications. This approach is inspired by ongoing work that uses execution scenarios to aid application development [23]. In this setting, the user draws a series of execution scenarios that characterise the intended behaviour of a concurrent system. If sufficient scenarios are given then the program code of the system can be inferred, with additional guidance from the user in discarding unwanted behaviour. A similar process could be applied to the Channel Ambient calculus for the design of mobile applications, as illustrated in Fig. 3 and Fig. 4.
References


A The Channel Ambient Calculus

\[ P, Q, R ::= \begin{align*}
0 & \quad \text{Null} \\
\parallel & \quad \text{Parallel} \\
\bar{\nu} n & \quad \text{Restriction} \\
\alpha & \quad \text{Action} \\
\! \alpha & \quad \text{Replication}
\end{align*} \]

\[ \alpha ::= a \cdot x \langle v \rangle \quad \text{Sibling Output} \]

\[ x \uparrow \langle v \rangle \quad \text{Parent Output} \]

\[ x( u ) \quad \text{Internal Input} \]

\[ x \uparrow ( u ) \quad \text{External Input} \]

\[ \text{in} \ a \cdot x \quad \text{Enter} \]

\[ \text{out} \ x \quad \text{Leave} \]

\[ \text{in} \ x \quad \text{Accept} \]

\[ \text{out} \ x \quad \text{Release} \]

Definition 3. Syntax of CA

\[ a \cdot b \cdot \langle v \rangle. P \mid Q \mid ( Q' \mid R ) \rightarrow a \cdot P' \mid Q \mid R \quad (1) \]

\[ a \cdot x \langle v \rangle. P \mid Q \mid ( Q' \mid R ) \rightarrow a \cdot P' \mid Q \mid R \quad (2) \]

\[ b \cdot \text{in} \ b \cdot x \cdot P \mid Q \mid \text{in} \ x \cdot Q \mid R \rightarrow b \cdot Q \mid Q' \mid a \cdot P' \mid R' \quad (3) \]

\[ b \cdot \text{out} \ x \cdot P \mid Q \mid \text{out} \ x \cdot Q \mid R \rightarrow b \cdot Q \mid Q' \mid a \cdot P' \mid R' \quad (4) \]

\[ P \rightarrow P' \Rightarrow P \mid Q \rightarrow P' \mid Q \quad (5) \]

\[ P \rightarrow P' \Rightarrow \nu n P \rightarrow \nu n P' \quad (6) \]

\[ P \rightarrow P' \Rightarrow a \cdot P \rightarrow a \cdot P' \quad (7) \]

\[ Q \equiv P \rightarrow P' \equiv Q' \Rightarrow Q \equiv Q' \quad (8) \]

Definition 4. Reduction in CA

\[ 0 \mid P \equiv P \quad (9) \]

\[ P \mid Q \equiv Q \mid P \quad (10) \]

\[ \nu n ( Q \mid R ) \equiv ( P \mid Q ) \mid R \quad (11) \]

\[ \nu n 0 \equiv 0 \quad (12) \]

\[ \nu n \nu m P \equiv \nu m \nu n P \quad (13) \]

\[ \text{fn}(\alpha) \not\subseteq \text{bn}(\alpha) \Rightarrow \nu \alpha \cdot P \equiv \alpha \cdot ( P \mid \nu \alpha \cdot P ) \quad (14) \]

\[ n \not\subseteq \text{fn}(Q) \Rightarrow ( \nu n P ) \mid Q \equiv \nu n ( P \mid Q ) \quad (15) \]

\[ a \neq n \Rightarrow a \cdot \nu n P \equiv \nu n a \cdot P \quad (16) \]

\[ \nu a \cdot 0 \equiv 0 \quad (17) \]

Definition 5. Structural Congruence in CA
\[ P \equiv P \quad (9) \quad \text{fn}(\alpha) \not\subseteq \text{ln}(\alpha) \Rightarrow \nu_{z} P \equiv \alpha.(P | \alpha.P) \quad (14) \]

\[ P \mid Q \equiv Q \mid P \quad (10) \quad n \not\subseteq \text{ln}(Q) \Rightarrow \quad P \mid Q \equiv P \mid Q \quad (15) \]

\[ P \mid Q \mid R \equiv P \mid Q \mid R \quad (11) \quad a \not\subseteq n \Rightarrow \quad P \equiv P \quad (16) \]

\[ n \not\subseteq \quad a \equiv a \quad (12) \quad 0 \equiv 0 \quad (17) \]

\[ m \not\subseteq \quad n \equiv n \quad (13) \]

**Definition 6.** Graphical Structural Congruence in CA

\[ z \not\subseteq \text{fn}(x, v, P) \Rightarrow x(v).P \equiv \nu_{z}(z,x(v),z(v) | z).P \]
\[ z \not\subseteq \text{fn}(a, x, v, P) \Rightarrow a/x(v).P \equiv \nu_{z}(a,x(v),z(v) | z).P \]

**Definition 7.** Syntax Abbreviations in CA

**Fig. 5.** Calculus Specification of the Resource Monitoring Application