A Coordination Model for Ad Hoc Mobile Systems

Marco Tulio Valente[1], Fernando Magno Pereira[2], Roberto da Silva Bigonha[2], and Mariza Andrade da Silva Bigonha[2]

[1] Department of Computer Science, Catholic University of Minas Gerais, Brazil
[2] Department of Computer Science, Federal University of Minas Gerais, Brazil
`mtov@pucminas.br`, {`fernandm,bigonha,mariza`}`@dcc.ufmg.br`

Abstract. The growing success of wireless ad hoc networks and portable hardware devices presents many interesting problems to system engineers. Particular, coordination is a challenging task, since ad hoc networks are characterized by very opportunistic connections and rapidly changing topologies. This paper presents a coordination model, called PeerSpaces, designed to overcome the shortcomings of traditional coordination models when used in ad hoc networks.

1 Introduction

With the advent of ad hoc networks, mobile devices can detach completely from the fixed infrastructure and establish transient and opportunistic connections with other devices that are in communication range. Designing applications on these dynamic and fluid networks presents many interesting problems. Particularly, coordination is a challenging task. Since a user may find itself in a different network at any moment, the services available to him change along the time. Thus, computation should not rely on any predefined and well known context. Specifically when operating in ad hoc mode, coordination should not assume the existence of any central authority, since the permanent availability of this node can not be granted. Communication should also be uncoupled in time and space, meaning that two communicating entities do not need to establish a direct connection to exchange data nor must know the identity of each other.

Recently, shared space coordination models, inspired by Linda [4], are being considered for communication, synchronization and service lookup in mobile computing systems. The generative communication paradigm introduced by Linda is based on the abstraction of a tuple space. Processes communicate by inserting, reading and removing ordered sequences of data from this space.

In traditional Linda systems, like TSpaces [8] and JavaSpaces [3], the tuple space is a centralized and global data structure that runs in a pre-defined service provider. In the base station scenario this server can easily be located in the fixed network. However, if operation in ad hoc mode is a requirement, the fixed infrastructure simply does not exist. This suggests that standard client/server implementations of Linda are not suitable to ad hoc scenarios, since they assume a tight coupling between client and servers and the permanent availability of the latter.

This paper formalizes our attempts to customize and adapt shared space coordination models to applications involving mobile devices with ad hoc network capabilities. The model formalized in the paper, called PeerSpaces, has primitives for local and remote communication, process mobility and service lookup. In order to answer the new requirements posed by ad hoc mobile computing systems, PeerSpaces departs from traditional client/server architectures and push towards a completely decentralized one. In the model, each node (or peer) has the same capabilities, acting as client, shared space provider and as router of messages. In order to provide support to operation in ad hoc mode, service lookup is distributed along the network and does not require any previous knowledge about its topology.

The paper is organized as follows. In Section 2 we informally present the PeerSpaces model, including its main design goals, concepts and primitives. In Section 3 we give the formal semantics of the model in terms of a small language derived from the π-calculus. Besides a precise specification of the model, the semantics presented in this section supports formal reasoning about applications built on PeerSpaces. Section 4 compares the model with similar efforts. Finally, Section 5 concludes the paper.

2 The PeerSpaces Model

The main concepts used in PeerSpaces are the following:

Hosts The model assumes that hosts are mobile devices. Each host has its own local tuple space and a running process. A host is written $h_g[\,P,\,T]$, where h is the name of the host, P is the process running in the host, T is its local tuple space, and g is the group of the host.

In PeerSpaces, the name of a host is different from the name of all other hosts. The model also assumes a infinite set H of possible host names.

Groups Hosts in the model are logically organized in groups. Each group has a name and can also contain subgroups, creating a tree structure. The group of a host is denoted by a tuple $\langle g_1, \ldots, g_n \rangle$, that specifies the path from the root group g_1 to the leaf group g_n where the host is located. For example, the tuple $\langle \texttt{pucminas}, \texttt{cs}, \texttt{proglab} \rangle$ denotes the set of hosts in the `proglab` group, which is a subgroup of the group `cs`, which is nested in the root group `pucminas`. Two groups can have the same name, as long they are not subgroups of the same group.

Network Mobile hosts in the model are connected by a wireless and ad hoc network. As usual in such networks, connectivity is transient and determined by the distance among hosts. Consequently, the topology of the network is continuously changing. In PeerSpaces, a network with hosts $h_1, h_2, \ldots, h_n$ is denoted by:

$$h_{1_{g_1}}[P_1,\, T_1] \mid h_{2_{g_2}}[P_2,\, T_2] \mid \ \ldots \ \mid h_{n_{g_n}}[P_n,\, T_n]\ ,\ E$$

where $g_1, g_2, \ldots, g_n$ are the group of the hosts and $E\colon H \times H$ is a relation representing connections among hosts. The presence of a pair (h_i, h_j) in E, denoted by $h_i \bowtie h_j$, indicates that host h_i is in communication range with host h_j. This relation is in continuous change to reflect reconfigurations in the network.

PeerSpaces also defines a set of primitives to assemble applications using the previous defined concepts. We spend the rest of this section describing such primitives.

Local Primitives The local tuple space of any host is accessed using the traditional **in** , **rd** and **out** primitives from Linda. Furthermore, there is a **chgrp** g primitive, used to change the group of the current host to the one specified by tuple g.

Process Mobility Processes in PeerSpaces are mobile in order to model the behavior of mobile agents. A mobile agent is a process that can move among sites carrying computation and accessing resources locally. In wireless environments, agents are a powerful design tool to overcome latency and to embed autonomous computations. In the model, the primitive **move** $h.P$ is used to move a process to node h, where its execution continues as P. If host h is not connected, the operation blocks until a connection to such host is established.

Remote Primitives Crucial to the scalability and efficiency of any coordination model for mobile computing systems is the design of the remote operations. Thus, from the beginning PeerSpaces departs from the idea of providing seamless access to a global and centralized space. Instead, there are primitives that operate in the remote space of a well-known host h: **out** h, v; **in** h, p, x and **rd** h, p, x, where v is a tuple, p is a pattern and x is a variable. These operations are merely remote implementations of the traditional Linda primitives and thus do not impact in the overall performance of the system.

As its local version, the remote **out** h, v primitive is asynchronous. The primitive is used when a process wants to leave a information to be consumed later in another host. In order to model its asynchronous behavior, the operation is executed in two steps. In the first step, a tag is added to the tuple v to indicate that it should be transfer as soon as possible to the destination host h. The tagged tuple, denoted by v_h, is then outputted in the local space of the host h' that requested the operation. In the second step, tuple v_h is transferred to the space of host h as soon as it is connected to h' and the tag is removed from the tuple. Since both steps are not atomic, while the tuple is "misplaced" in the source node it can be retrieved by an operation like **in** v_h. For example, this operation can be called by a garbage collector process in charge of reclaim tuples that are waiting for the connection of their destination host for a long time.

Lookup Primitive Without a lookup primitive the remote operations described above have little use, since a mobile host may not know in advance the name h of a service provider in its current network. Moreover, since the system is designed to support operation in ad hoc mode, the lookup service must not be centralized in a single host, but must be distributed along the federation of connected devices. In order to accomplish such requirements, there is in PeerSpaces the following primitive: **find** g, p. This primitive queries hosts in group g for tuples matching pattern p in a distributed way. All matching tuples found in group g are copied asynchronously to the local space of the host that has called the operation.

The semantics of PeerSpaces does not assume any specific routing protocol for propagation of lookup queries.

Continuous Queries Often it is useful to query a group of hosts for a resource and keep the query effective until such resource is available. In this way, a client does not need to periodically send lookup queries to detect new resources that may become available since the last query was issued. In PeerSpaces, lookup queries that remain active after their first execution are called continuous queries.

Continuous queries in PeerSpaces have a lifetime parameter, used to automatically garbage collect the query after its expiration. Continuous lookup queries are issued adding the lifetime t to the **find** primitive: **find** g, p, t. This primitive will search the hosts of group g for all currently available resources matching pattern p and for resources that may become available in t units of time after the query was issued.

3 Formal Semantics

The ultimate goal of our research is to deploy a coordination middleware for ad hoc mobile computing systems. In order to achieve this goal we have initially defined the formal semantics of PeerSpaces.

The formalization presented next uses an operational semantics based on the asynchronous π-calculus [6]. The π-calculus is a good basis as it provides a small, elegant and expressive concurrent programming language. The main departure from π in our semantics is the use of generative communication instead of channel-based communication.

Table 1 summarizes the syntax of our core language. We assume an infinite set H of names, used to name hosts and lookup queries. Meta-variables h and x range over H. Basic values, ranged over by v and g, consist of names and tuples. Tuples are ordered sequences of values $\langle v_1, \ldots, v_n \rangle$. A tuple space T is a multiset of tuples. We use the symbol $? \in H$ to denote the distinguished unspecified value.

A program is composed by the network N, the relation E and a global set of names X. The relation $E : H \times H$ represents the connectivity map of the network. The names used over several hosts in the system are recorded in the set X, ensuring their unicity. Each host h is member of a group g and has a running process P and a local tuple space T. Processes are ranged by P and Q. Similar

$$
\begin{aligned}
Prog &::= N \,,\, E \,,\, X \\
N &::= \varepsilon \;\mid\; H \mid N \\
H &::= h_g[\,P,\, T\,] \\
P &::= \mathbf{0} \;\mid\; P \mid Q \;\mid\; !\,P \;\mid\; (\nu\, x)\, P \;\mid\; \mathbf{out}\; v \;\mid \\
&\quad\;\; \mathbf{in}\; v, x.P \;\mid\; \mathbf{rd}\; v, x.P \;\mid\; \mathbf{find}\; g, p, t \;\mid \\
&\quad\;\; \mathbf{chgrp}\; g \;\mid\; \mathbf{move}\; h.P
\end{aligned}
$$

Table 1. Syntax

to the π-calculus, the simplest term of our language is the inert process $\mathbf{0}$, which denotes a process with no behavior at all. The term $P \mid Q$ denotes two processes running in parallel. The term $!\,P$ denotes a infinite number of copies of P, all running in parallel. The restriction operator $(\nu\, x)\, P$ ensures that x is a fresh and unguessable name in the scope of P. Similar to Linda, the primitive operations **out**, **in** and **rd** provide access to the local tuple space. Since the **out** operation is asynchronous it does not have a continuation P. The same happens to the **find** and **chgrp** primitives. We assume that non-continuous lookup queries can be simulated by defining the lifetime equal to zero. Finally, the **move** operation simulates the behavior of single thread mobile agents.

The operational semantics of our calculus is summarized in Tables 2 and 3. The semantics is defined in terms of a reduction relation $\rightarrow$, a structure congruence $\equiv$ between processes and a set of pattern matching rules.

Table 2 summarizes the core language semantics, which is basically Linda with multiple tuple spaces. A reduction $N \,,\, E \,,\, X \;\rightarrow\; N' \,,\, E' \,,\, X'$ defines how the configuration $N \,,\, E \,,\, X$ reduces in a single step computation to $N' \,,\, E' \,,\, X'$. Initially, there are three reduction rules describing the effects on the configuration of each standard Linda primitive. The output operation, **out** v, asynchronously deposits a tuple in the local space (rule L1). The input, **in** $v, x.P$, and read, **rd** $v, x.P$, operations try to locate a tuple v' that matches v (rules L2 and L3). If one is found, free occurrences of x are substituted for v' in P, denoted as $P\{v'/x\}$. In the case of the input, the tuple is removed from the space.

The next set of rules defines a structural congruence relation $\equiv$ between processes (SC1 to SC7) and hosts (SC8 to SC10). As in the π-calculus, such rules define how processes can be syntactically rearranged in order to allow the application of reductions. In such rules, we write $fn(P)$ to denote the set of free names in process P. The definition of pattern matching, written $v \leq v'$, allows for recursive tuple matching. Values match only if they are equal or if the unspecified value occurs on the left hand side.

Table 3 extends the core language with the primitives proposed in PeerSpaces. The **find** g', p, t operation deposits a tuple representing a service lookup query in the local space (rule P1). Such query is a tuple in the format $\langle k, g', p, t, h \rangle$, where k is a fresh name that identifies the query, g' defines the group where the query

Reductions

Linda Primitives

$$h_g[\,\mathbf{out}\ v \mid P,\, T] \mid N\,,\, E\,,\, X \ \rightarrow\ h_g[\,P,\, v \cup T] \mid N\,,\, E\,,\, X \qquad \text{(L1)}$$

$$h_g[\,\mathbf{in}\ v, x.P \mid Q,\, v' \cup T] \mid N\,,\, E\,,\, X \ \rightarrow\ h_g[\,P\{v'/x\} \mid Q,\, T] \mid N\,,\, E\,,\, X \qquad \text{(L2)}$$

$$h_g[\,\mathbf{rd}\ v, x.P \mid Q,\, v' \cup T] \mid N\,,\, E\,,\, X \ \rightarrow\ h_g[\,P\{v'/x\} \mid Q,\, v' \cup T] \mid N\,,\, E\,,\, X \qquad \text{(L3)}$$

The rules are subjected to the following side conditions:
(L2) if $v \leq v'$
(L3) if $v \leq v'$

Structural Congruence Rules

$P \mid Q \equiv Q \mid P$	(SC1)	$(\nu\, x)\,(\nu\, y)\,P \equiv (\nu\, y)\,(\nu\, x)\,P$	(SC5)
$!\,P \equiv P \mid\, !\,P$	(SC2)	$P \equiv Q \Rightarrow (\nu\, x)\,P \equiv (\nu\, x)\,Q$	(SC6)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(SC3)	$(\nu\, x)\,(P \mid Q) \equiv P \mid (\nu\, x)\,Q$	(SC7)
$P \mid \mathbf{0} \equiv P$	(SC4)		

$$P \equiv Q \Rightarrow h_g[\,P,\, T] \equiv h_g[\,Q,\, T] \qquad \text{(SC8)}$$

$$h_g[\,(\nu\, x)\,P,\, T] \mid N\,,\, E\,,\, X \equiv h_g[\,P,\, T] \mid N\,,\, E\,,\, x \cup X \qquad \text{(SC9)}$$

$$h_g[\,P,\, T] \mid N\,,\, E\,,\, X \equiv N \mid h_g[\,P,\, T]\,,\, E\,,\, X \qquad \text{(SC10)}$$

The rules are subjected to the following side conditions:
(SC7) if $x \notin fn(P)$
(SC9) if $x \neq h,\ x \notin fn(N),\ x \notin X$

Pattern Matching Rules

$$v \leq v \qquad ? \leq\ v \qquad \frac{v_1 \leq v_1' \ \ldots\ v_n \leq v_n'}{\langle v_1\ \ldots\ v_n\rangle \leq \langle v_1'\ \ldots\ v_n'\rangle}$$

Table 2. Core Language Operational Semantics

will be performed, p is a pattern for the desired service, t is the lifetime and h is the name of the current host. The operation **chgrp** g just changes the group of the current host to the one specified by tuple g (rule P2). If such group does not exist, it is created. The **move** $h'.P$ operation changes the location of the continuation process P to host h' if this host is connected (rule P3). Otherwise, the operation remains blocked until the engagement of h'.

Reduction rule Q1 defines how lookup queries are propagated in the network. Basically, any host that holds a query $\langle k, g'', p, t, h\rangle$ can propagate it to a connected host h' in group g', if g'' matches g' and the query is not yet present in h'. If such conditions are satisfied, the query is inserted in the local space of h' and a process P'' is added in parallel with the other processes running in this host. This process continuously read tuples matching the pattern p and then use

Reductions

PeerSpaces Primitives

$$h_g[\,\textbf{find}\ g',p,t \mid P,T] \mid N\,,E\,,X \;\rightarrow\; h_g[(\nu\, k)\,P,\,\langle k,g',p,t,h\rangle \cup T] \mid N\,,E\,,X \quad \text{(P1)}$$

$$h_g[\,\textbf{chgrp}\ g' \mid P,T] \mid N\,,E\,,X \;\rightarrow\; h_{g'}[P,T] \mid N\,,E\,,X \quad \text{(P2)}$$

$$\begin{aligned} &h_g[\,\textbf{move}\ h'.P \mid Q,T] \mid h'_{g'}[P',T'] \mid N\,,E\,,X \;\rightarrow \\ &\quad h_g[Q,T] \mid h'_{g'}[P \mid P',T'] \mid N\,,E\,,X \end{aligned} \quad \text{(P3)}$$

Query Propagation

$$\begin{aligned} &h_g[P,\,\langle k,g'',p,t,h\rangle \cup T] \mid h'_{g'}[P',T'] \mid N\,,E\,,X \;\rightarrow \\ &\quad h_g[P,\,\langle k,g'',p,t,h\rangle \cup T] \mid h'_{g'}[P' \mid P'',\,\langle k,g'',p,t,h\rangle \cup T'] \mid N\,,E\,,X \end{aligned} \quad \text{(Q1)}$$

Network Reconfiguration

$$\frac{E \;\Rightarrow\; E'}{N\,,E\,,X \;\rightarrow\; N\,,E'\,,X} \quad \text{(N1)}$$

The rules are subjected to the following side conditions:
(P3) if $h \bowtie h'$
(Q1) if $(h \bowtie h') \wedge (g'' \preceq g') \wedge (\langle k,g'',p,t,h\rangle \notin T') \wedge P'' = !\,(\textbf{rd}\ p,x.\textbf{out}\ h,x)$

Group Matching Rule

$$\frac{g_1 = g_1' \;\ldots\; g_n = g_n'}{\langle g_1 \ldots g_n\rangle \preceq \langle g_1' \ldots g_n' \ldots\; g_m'\rangle}$$

Table 3. PeerSpaces Operational Semantics

a remote output operation to send the results to the local space of the host h that has issued the query.

The last reduction rule introduces a new type of reduction $\Rightarrow$ used to describe reconfigurations in the network and consequently in the connectivity relation E. Basically, this rule dictates that changes in E should be propagated to the current configuration. However, we left $\Rightarrow$ reductions unspecified in the semantics, since they are dependent on the physical location of each host and on technological parameters of the subjacent network.

There is also a special pattern matching rule for group names. Two groups g and g' matches, written $g \preceq g'$ if all subgroups in g are equal to equivalent subgroups in g', which can also have extra nested subgroups.

4 Related Work

Many characteristics of PeerSpaces have been inspired in file sharing applications popular in the Internet. Particularly, the peer to peer network created by Gnutella [5] over the fixed Internet presents many properties that are interesting in mobile settings, like absence of centralized control, self-organization and adaptation to failures. PeerSpaces is an effort to transport and adapt such characteristics to mobile computing systems. This explains the choice of Linda shared spaces as the prime coordination infrastructure for PeerSpaces.

Lime [7] introduces the notion of transiently shared data space to Linda. In the model, each mobile host has its own tuple space. The contents of the local spaces of connected hosts are transparently merged by the middleware creating the illusion of a global and virtual data space. Applications in Lime perceive the effects of mobility by atomic changes in the contents of this virtual space. The scalability and performance weakness of Lime have motivated the proposal of CoreLime [2], where in name of simplicity and scalability the idea of transiently shared spaces is restricted to the set of mobile agents running in a host. Another work proposing an alternative semantics to the notion of transiently shared spaces is [1].

5 Conclusions

In this paper we have presented and formalized PeerSpaces, a coordination model for mobile computing systems. The model was designed to overcome the main shortcoming of shared spaces coordination models when used in ad hoc wireless networks – the strict reliance on the traditional client/server architecture – while preserving the main strengths of such models – the asynchronous and uncoupled style of communication. PeerSpaces can be used as the building block of ad hoc mobile systems like file sharing, groupware, mobile commerce and message systems.

Acknowledgments We would like to thank Jan Vitek and Bogdan Carbunar for the discussions that led to this paper.

References

1. N. Busi and G. Zavattaro. Some thoughts on transiently shared tuple spaces. In *Workshop on Software Engineering and Mobility*, May 2001.
2. B. Carbunar, M. T. Valente, and J. Vitek. Corelime a coordination model for mobile agents. In *International Workshop on Concurrency and Coordination*, volume 54 of *Electronic Notes on Theoretical Computer Science*. Elsevier Science, July 2001.
3. E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley, 1999.
4. D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, Jan. 1985.
5. Gnutella Home Page. `http://gnutella.wego.com`.
6. R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.
7. A. L. Murphy, G. P. Picco, and G.-C. Roman. Lime: A middleware for physical and logical mobility. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, May 2001.
8. P. Wycko, S. W. McLaughry, T. J. Lehman, and D. A. Ford. TSpaces. *IBM Systems Journal*, 37(3):454–474, Aug. 1998.