*T*uples *O*n *T*he *A*ir: a Middleware for Context-Aware Computing in Dynamic Networks

Marco Mamei[1,2], Franco Zambonelli[2], Letizia Leonardi[1]

[1]*DII – Università di Modena e Reggio Emilia –Via Vignolese 905 – Modena – ITALY*
[2]*DISMI – Università di Modena e Reggio Emilia – Via Allegri 13 – Reggio Emilia – ITALY*
{ mamei.marco, franco.zambonelli, letizia.leonardi }@unimo.it

Abstract

We present TOTA ("Tuples On The Air"), a novel middleware for supporting adaptive context-aware application in dynamic network scenarios. The key idea in TOTA is to rely on spatially distributed tuples for both representing contextual information and supporting uncoupled and adaptive interactions between application components. The middleware propagates tuples across a network on the basis of application-specific patterns and adaptively re-shapes the resulting distributed structures accordingly to changes in the network scenario. Application components can locally "sense" these structures and exploit them to acquire contextual information and carry on complex coordination activities in an adaptive way. Several examples show the effectiveness of the TOTA approach.

1. Introduction

Scenarios such as mobile [Bro98, Mam02], peer-to-peer [RowD01, Rat01], and pervasive computing [Est02], call for novel approaches to support and facilitate the development and execution of distributed applications. In particular: *(i)* the activities of software components are often related to the environment in which the hosting device situates, requiring components to execute in a *context-aware* way by dynamically acquiring information about the surrounding environment. *(ii)* The openness and dynamics of the scenario, due to mobile and ephemeral nodes, require applications to be *adaptive*, i.e., capable of dealing with such dynamics in an unsupervised way. *(iii) simplicity* of applications and supporting infrastructures, always appreciated, is further motivated by the presence of resource- and power-limited devices.

The above issues do not find effective solutions in traditional approaches to distributed computing. There, distributed software components are usually assumed as fixed and connected with each other via reliable network connections. Consequently: *(i)* application components, per se, are provided with either no contextual information at all or with only low-expressive information (e.g., raw local data or simple events), that they have to explicitly acquire via external services; *(ii)* components are typically strictly coupled in their interactions (e.g., as in message-passing models), thus making it difficult to promote and support adaptive interoperability and coordination without the reliable and continuous support of external services; *(iii)* the results is usually in both heavyweight supporting infrastructures and in an increase of application complexity.

TOTA ("Tuples On The Air") is a novel middleware infrastructure explicitly conceived as a support for distributed computing in dynamic network scenarios. The key objectives of TOTA, only partially achieved by similar proposals [PicMR01, BabMR02, RomJH02] are: *(i)* to promote uncoupled and adaptive interactions by locally providing application components with simple yet expressive contextual information; and *(ii)* to actively support adaptivity by discharging application components from the duty of dealing with network and application dynamics. To this end, TOTA relies on spatially distributed tuples, to be injected in the network and propagated accordingly to application-specific patterns. Tuple propagation patterns are dynamically re-shaped by the TOTA middleware to implicitly reflect network and applications dynamics, as well as to reflect the evolution of coordination activities. Application components have simply to locally "sense" tuples to acquire contextual information, exchange information with each other, and adaptively orchestrate their coordination activities.

2. Motivations and Related Works

A number of recent proposals addresses the limitations of traditional computing models and middleware, with the general goal of creating supporting environments for the development of adaptive, context-aware distributed applications, suitable for dynamic network scenarios.

Some approaches propose relying on shared data structures as the basis for uncoupled interactions and

context-awareness. For instance, the Lime [PicMR01] and the XMIDDLE [MasCE01] middleware exploit tuple spaces and XML trees, respectively, as the basis for interaction in dynamic network scenario. Each device in the network owns a private data structure (e.g., a private tuple space or a portion of an XML tree). Upon connection with other devices in a network, the privately owned data structures can merge together accordingly to specific policies, to be used as a common interaction space to exchange contextual information and coordinate with each other. By letting the shared data space mediate and uncouple all interactions, and by not relying on any centralized service, these approaches suit well the dynamics of wireless open networks. However, the acquired information is typically strictly local (deriving from the sharing of data among directly connected nodes) and is of no support in a acquiring a more global perspective and in achieving complex distributed coordination patterns over a possibly large network.

The approach proposed in [RomJH02] is explicitly targeted at facilitating the acquisition of contextual information in dynamic scenarios (i.e., MANETs). There, each node in the network can specify an interest for some contextual information, together with the scope of that interest (e.g. all gas stations within 10 miles). The middleware is then in charge to build a distributed data structure (i.e., a shortest path tree) spanning all the peers within the scope of the interest. This data structure will then be used to route back to the node the contextual information that can be collected from the other nodes in the network. Such an approach may be very powerful for locally gathering contextual information without strict locality constraint and without requiring any centralized service (all nodes cooperate to provide the information in a distributed way). However, the proposal lacks flexibility, in that is purely focused on information gathering, and pays little or no attention to the problem of coordinating the activities of application components: once components have gathered contextual information they are left on their own, without being supported in their coordination activities. For instance, even if components can dynamically acquire reciprocal knowledge, they may be thereafter forced to interact in a direct way to coordinate their activities with each other.

Anthill [BabM02] is a framework built to support design and development of adaptive peer-to-peer applications, that exploits an analogy with biological adaptive systems [Bon99]. Anthill consists of a dynamic network of peer nodes, each one provided with a local tuple space ("nest"), in which distributed mobile components ("ants") can travel and can indirectly interact and cooperate with each other by leaving and retrieving tuples ("pheromones") in the distributed tuple spaces. The key objective of Anthill is to build robust and adaptive networks of peer-to-peer services (e.g., file sharing) by exploiting the capabilities of ants to re-shape their activity patterns accordingly to the changes in the network structure. Although we definitely find the idea interesting and promising, a more general flexible approach would be needed to support – other than adaptive resource sharing – adaptive coordination in distributed applications.

3. An Overview of TOTA

The driving objective of our approach is to address together the two requirements introduced at the beginning of the previous section (context-awareness and uncoupled and adaptive interactions), by exploiting a unified and flexible mechanism to deal with both context representation and components' interaction, and thus also leading to simpler, and lighter to be supported, applications.

In TOTA, we propose relying on distributed tuples for both representing contextual information and enabling uncoupled interaction among distributed application components. Unlike traditional shared data space models, tuples are not associated to a specific node (or to a specific data space) of the network. Instead, tuples are injected in the network and can autonomously propagate and diffuse in the network accordingly to a specified pattern (see Figure 1). Thus, TOTA tuples form a sort of spatially distributed data structure able to express not only messages to be transmitted between application components but, more generally, some contextual information on the distributed environment.

To support this idea, TOTA is composed by a peer-to-peer network of possibly mobile nodes, each running a local version of the TOTA middleware. Each TOTA node holds references to a limited set of neighboring nodes. The structure of the network, as determined by the neighborhood relations, is automatically maintained and updated by the nodes to support dynamic changes, whether due to nodes' mobility or to nodes' failures.

Upon the distributed space identified by the dynamic network of TOTA nodes, each component is capable of locally storing tuples and letting them diffuse through the network. Tuples are injected in the system from a particular node, and spread hop-by-hop accordingly to their propagation rule. In fact, a TOTA tuple is defined in terms of a "content", and a "propagation rule".

$$T=(C,P)$$

The content C is an ordered set of typed fields representing the information carried on by the tuple. The propagation rule P determines how the tuple should be distributed and propagated in the network. This includes determining the "scope" of the tuple (i.e. the distance at which such tuple should be propagated and possibly the

spatial direction of propagation) and how such propagation can be affected by the presence or the absence of other tuples in the system. In addition, the propagation rules can determine how tuple's content should change while it is propagated. Tuples are not necessarily distributed replicas: by assuming different values in different nodes, tuples can be effectively used to build a distributed overlay data structure expressing some kind of contextual and spatial information. On a different perspective, we can say that TOTA enrich a network with a notion of space. A tuple incrementing one of its fields as it gets propagated identifies a sort of "structure of space" defining the network distances from the source.

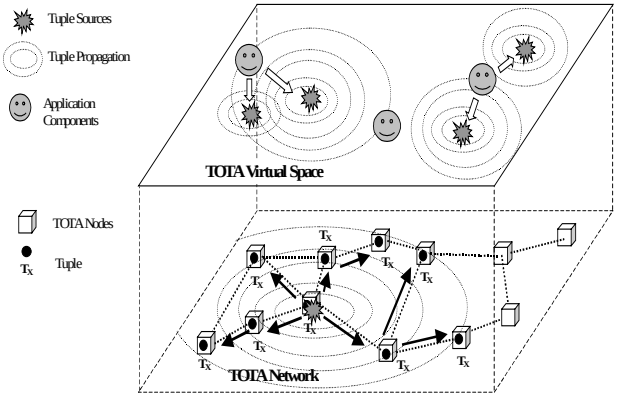

Figure 1: **The TOTA scenario: application components live in an environment in which they can inject tuples that autonomously propagate and sense tuples present in their local neighborhood. This is realized by a peer-to-peer network where tuples propagates by means of a multi-hop mechanism.**

By relying on data acquired by proper physical localization devices, like GPS systems or Wi-Fi triangulation, tuples CAN provide a structure of space based on the actual physical location of devices and thus enabling a tuple to be propagated, say, at most for 10 meters from its source. Taking this approach to the extreme, one could think at mapping the peers of a TOTA network in any sort of virtual overlay space [Rat01], and propagating tuples accordingly to the virtual space topology.

The spatial structures induced by tuples propagation must be maintained coherent despite network dynamism. To this end, the TOTA middleware supports tuples propagation actively and adaptively: by constantly monitoring the network local topology and the income of new tuples, the middleware automatically re-propagates tuples as soon as appropriate conditions occur. For instance, when new nodes get in touch with a network, TOTA automatically checks the propagation rules of the already stored tuples and eventually propagates the tuples to the new nodes. Similarly, when the topology changes due to nodes' movements, the distributed tuple structure automatically changes to reflect the new topology.

Form the application components' point of view, executing and interacting basically reduces to inject tuples, perceive local tuples, and act accordingly to some application-specific policy. Software components on a TOTA node can inject new tuples in the network, defining their content and their propagation rule. They have full access to the local content of the middleware (i.e., of the local tuple space), and can query the local tuple space – via a pattern-matching mechanism – to check for the local presence of specific tuples. In addition, components can be notified of locally occurring events (i.e., changes in tuple space content and in the structure of the network neighborhood). In TOTA there is not any primitive notion of distributed query. Still, it is possible for a node inject a tuple in the network and have such distributed tuple be interpreted as a query at the application-level, by having other components in the network react to the income of such tuple, i.e., by injecting a reply tuple propagating towards the enquiring node.

The overall resulting scenario is that of applications whose components: *(i)* can influence the TOTA space by propagating application-specific tuples; *(ii)* execute by being influenced in both their internal and coordination activities by the locally sensed tuples; and *(iii)* implicitly tune their activities to reflect network dynamics, as enabled by the automatic re-shaping of tuples' distributions of the TOTA middleware. To some extent, TOTA mimics the way electromagnetic and gravitational fields propagate in space and influence the activities (i.e., the movements) of particles locally sensing such fields.

4. TOTA Middleware

4.1. TOTA Architecture

As introduced in the previous section, a network of possibly mobile nodes running each one a TOTA middleware constitutes the scenario we consider. Each TOTA node holds references to neighboring nodes and it can communicate directly only with them. While in an ad-hoc network scenario it is rather easy to identify the node's neighborhood with the range of the wireless link, in a wired scenario like the Internet is less trivial. We imagine however that in such a case the term is not related to the real reachability of a node, but rather on its addressability (a node can communicate directly with another only if it knows other node's IP address). This means that at the very bottom of the TOTA middleware

there is a system to continuously detect neighboring nodes and to store them in an appropriate list. In a MANET scenario this system is directly connected to the wireless network and detects in-range nodes. In a wired scenario, like the Internet, it can start an expanding-ring search for other TOTA nodes, or it can simply query a central repository (e.g. a known web-site) and download a list of TOTA nodes' IP addresses.

Other than maintaining the TOTA network each middleware is in charge to store, propagate and keep updated the tuples' structure. To achieve this task TOTA needs a mean to uniquely identify tuples in the system in order for example to know weather a particular tuple has been already propagated in a node or not. Tuple's content cannot be used for this purpose, because the content is likely to change during the propagation process. To this end, each tuple will be marked with an *id* (invisible at the application level) that will be used by TOTA during tuples' propagation and update to trace the tuple. Tuples' *id* is generated by combining a unique number relative to each node (i.e., the MAC address) together with a progressive counter for all the tuples injected by the node.

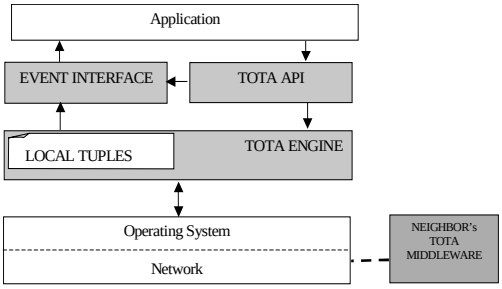

Figure 2: **The Architecture of TOTA**

From the architecture point of view, the TOTA middleware is constituted by three main parts (see Figure 2). The TOTA API, described in detail in the next subsection, is the main interface between the application and the middleware. It provides functionalities to let the application inject new tuples in the system, to access the local tuple space, or to place subscriptions in the event interface. The EVENT INTERFACE is the component in charge of asynchronously notifying the application about the income of a new tuple or about the fact a new node has been connected/disconnected to the node's neighborhood. The TOTA ENGINE is the core of TOTA: it is in charge of maintaining the TOTA network by storing the references to neighboring nodes and to manage tuples' propagation by opening communication sockets to send and receive tuples. This component is in charge of sending tuples injected by the application level and to apply the propagation rule of received tuples re-propagating them accordingly. Finally this component monitors network reconfiguration and the income of new tuples and updates and re-propagates already stored tuples to maintain tuples' structure coherency.

4.2. Implementation

From an implementation point of view, we developed a first prototype of TOTA running on Laptops and on Compaq IPAQs equipped with 802.11b and Personal Java. IPAQ connects locally in the MANET mode (i.e. without requiring access points) creating the skeleton of the TOTA network. Tuples are being propagated through multicast sockets to all the nodes in the one-hop neighbor. The use of multicast sockets has been chosen to improve the communication speed by avoiding 802.11b unicast handshake. By considering the way in which tuples are propagated, TOTA is very well suited for this kind of broadcast communication. We think that this is a very important feature, because it will allow in the future implementing TOTA also on really simple devices (e.g. micro sensors) that cannot be provided with sophisticate (unicast enabled) communication mechanisms. Other than this communication mechanism, at the core of the TOTA middleware there is a simple event-based engine, that monitors network reconfigurations and the income of new tuples and react either by triggering propagation of already stored tuples or by generating an event directed to the event interface.

Actually we own only a dozen of IPAQs and laptops on which to run the system. Since the effective testing of TOTA would require a larger number of devices, we have implemented a graphic emulator to analyze TOTA behavior in presence of hundreds of nodes. The emulator, developed in Java, enables examining TOTA behavior in a MANET scenario, in which nodes topology can be rearranged dynamically either by a drag and drop user interface or by autonomous nodes' movements (see a snapshot in Figure 3).

4.3. The TOTA API

TOTA is provided with a simple set of primitive operations to interact with the middleware. The main operation:

```
public void inject (Tuple tuple)
```

is used to inject the tuple passed as an argument in the TOTA network. Once injected the tuple starts propagating accordingly to its propagation rule (embedded in the tuple definition – see later on).

The following two primitives give access to the local list of stored tuples:

```
public ArrayList read (Tuple template)
public ArrayList delete (Tuple template)
```

The read primitive accesses the local TOTA tuple space and returns a collection of the tuples locally present in the tuple space and matching the template tuple passed as parameter. The delete primitive extracts from the local middleware all the tuples matching the template and returns them to the invoking component.

In addition, two primitives are defined to handle events:

```
public void subscribe (Tuple template,
                       String reaction)
public void unsubscribe (Tuple template)
```

These primitives rely on the fact that any event occurring in TOTA (including: arrivals of new tuples, connections and disconnections of nodes) can be represented as a tuple. Thus, the subscribe primitive associates the execution of a reaction method in the component (reaction's name is specified as second parameter) in response to the occurrence of events matching the template tuple passed as first parameter. The unsubscribe primitives removes matching subscriptions.

Other than this API, we must define a suitable model for tuples' programming. Being implemented in Java, TOTA relies on an object oriented tuple representation: the system is preset with an **abstract class Tuple** that provides a propagate method implementing – with the support of the middleware – a general-purpose breadth first, expanding-ring propagation. In this abstract class, four abstract methods are defined to control tuple propagation, its content update, and its behavior. These methods must be implemented when subclassing from the abstract class Tuple to create and instantiate actual tuples, so as to realize specific propagation patterns (the data fields of a tuple are instead represented by the internal attributes). This approach is very handy when programming new tuples, because it does not require to re-implement every time the propagation mechanism from scratch, but it allows customizing the same breadth first, expanding ring propagation for different purposes.

In general, the definition of the methods in a tuple class allows instances of the class to follow any needed propagation pattern. Among the others: flooding the network, propagating in a specific direction or within a confined spatial area, propagating without being stored in the propagation nodes, propagating by deleting/modifying specific tuples in the propagation nodes (this can be used to supply the lack of a "delete" primitive in the API), adapting the propagation pattern depending on data (i.e., on the value of some tuples) found in the propagation nodes. More details on the TOTA programming model, with also code examples, can be found in a companion paper [MamZL02].

5. Application examples

TOTA can be exploited to solve several problems typical of dynamic network scenarios, by simply implementing different tuples and propagation rules.

5.1. Routing on Mobile ad Hoc Networks

In wireless ad hoc networks, a major challenge lies in the design of routing mechanisms. In the last decade, a number of ad hoc network routing protocols have been proposed, which usually dynamically build a sort of routing overlay structure by flooding the network and then exploit this overlaid structure for a much finer routing. Far from proposing a new routing protocol, we will show in this section how the basic mechanism of creating a routing overlay structure and the associated routing mechanism (similar to the ones already proposed in the area) can be effectively done within the TOTA model. The basic routing algorithm we will try to implement is the following [Poo00]: when a node X wants to send a message to a node Y it injects a tuple representing the message to be sent, and a tuple used to create an overlay routing structure, for further use.

The tuple used to create the overlay structure can be described as follows:

C=("structure", nodeName, hopCount)

P=(propagate to all the nodes, increasing hopCount by one at every hop)

The tuple used to convey the message will be:

C=("message", sender,receiver,message)

P=(if a "structure" tuple having my same receiver can be found in the local node, follow downhill its hopCount, otherwise propagate to all the nodes)

This routing algorithm is very simple: *structure* tuples create an overlay structure so that a *message* tuple following downhill a *structure* tuple's *hopCount* can reach the node that created that particular structure. In all situations in which such information is absent, the routing simply reduces to flooding the network. Although its simplicity, this model captures the basic underling model of several different MANET routing protocols [Bro98]. The basic mechanism described in this section (tuples defining a structure to be exploited by other tuples' propagation) is fundamental in the TOTA approach and provides a great flexibility. For example it allows TOTA to realize systems providing content-based routing in the Internet peer-to-peer scenario, such as CAN [Rat01] and Pastry [RowD01].

5.2. Gathering Information

Building on the previous routing mechanism, another application that can be easily realized upon TOTA is a system to enable components to collect information in a

dynamic network. Basically, we can imagine that in an environment are located information nodes (e.g., sensors) providing some useful information, and mobile devices carried by users in need of collecting such information. To model such an application we can envision two very simple solutions.

As a first solution, we can have each node propagate a tuple having as a content *C* the information to be made available, as well as its location, and a value specifying the distance of the tuple from the node itself:

C= (description, location, distance)

P=(propagate to all peers hop by hop, increasing the "distance" field by one at every hop)

Then, any device, by simply checking its local TOTA tuple space, can acquire all the information propagated in the environment. Moreover, by following backwards the tuple up to its source, can easily reach the information's source without having to rely on any a priori global information about where sensors are located.

As an alternative solution, we can consider gathering contextual information by exploiting a request/response mechanism: user devices can inject tuples describing the information they are looking for. Information nodes, by their side, can be programmed to sense these tuples and respond accordingly. In particular, they can subscribe to the income of query tuples to which they can respond and react to such events by injecting an answer tuple. Basically the communication would proceed by following the routing algorithm described in the previous section (i.e. query tuples create a structure to be used by answer tuples to reach the enquiring device). Also, in that way, we achieve in TOTA the same functionalities proposed in [RomJH02].

Finally, it is rather easy to see that, by properly shaping tuples, we can achieve patterns of interactions analogous to the ones promoted by those middleware relying on virtually shared data structures [PicMR01, MasCE01].

5.3. Motion Coordination

To show the capability of achieving globally coordinated behaviors with TOTA, we focus on a specific instance of the general problem of motion coordination (i.e., having a group of components move in an environment in a coordinated way). Motion coordination has been widely studied in several research areas: robotics, simulations, pervasive computing, multi agent systems, etc. [KubCH01, Mam02]. Here we will consider the problem of letting a group of mobile components (e.g., users with a PDA or robots) move maintaining a specified distance from each other. To this end, we can take inspiration from the mechanism used by birds to flock [Bon99]: flocks of birds move in a coordinated way by simply trying to maintain a specified distance from the nearest birds and by match nearby birds' velocity. To implement this in TOTA, and let components maintain a specified distance *X* from each other, each component can generate a tuple *T=(C,P)* with following characteristics:

C= (FLOCK, nodeName,val)

P= ("val" is initialized at X, propagate to all the nodes decreasing by one in the first X hops, then increasing "val" by one for all the further hops)

Thus creating a distributed data structure in which the *val* field assumes the minimal value at specific distance from the source (e.g., *X* hops). To coordinate movements, components have simply to locally perceive the generated tuples, and to follow downhill the gradient of the *val* fields. The result is a globally coordinated movement in which components maintain an almost regular grid formation by clustering in each other *val* fields' minima.

Figure 3 shows a snapshot of the testing of this motion coordination patterns in the TOTA emulator.

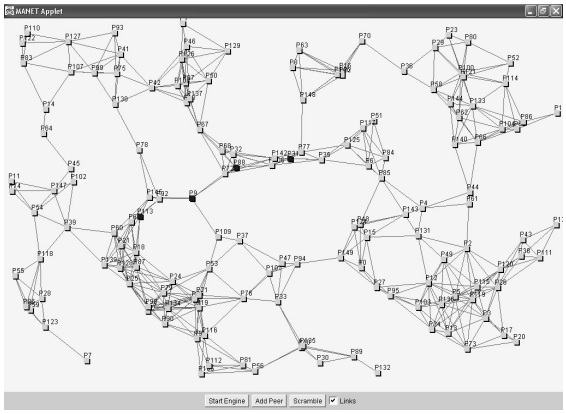

Figure 3**. Flocking in the TOTA Emulator. Cubes are the nodes of a mobile ad-hoc network, with arcs connecting nodes in range with each other. Black cubes are involved in flocking, moving by preserving a specified distance from each other.**

6. Conclusions and Future Works

The TOTA middleware, by relying on distributed tuples to be propagated over a network as sorts of electromagnetic fields, provides an effective support to support distributed applications in dynamic network scenarios, as we have tried to shown via several application examples.

Several issues are still to be solved for our first prototype implementation to definitely fulfill its promises. First, we must fully specify the operational semantics of TOTA operations, to ensure, e.g., their stability and the absence of critical races. Second, we must compulsory integrate proper access control model to

rule accesses to distributed tuples and their updates. Third, performance evaluations are needed to test the limits of usability and the scalability of TOTA by quantifying the TOTA delays in updating the tuples' distributed structures in response to dynamic changes.

References

[BabM02] O. Babaoglu, H. Meling, A. Montresor, "Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems", 22th International Conference on Distributed Computing Systems (ICDCS '02), Vienna (A), July 2002.

[Bon99] E. Bonabeau, M. Dorigo, G. Theraulaz, "Swarm Intelligence", Oxford University Press, 1999.

[Bro98] J. Broch, D. Maltz, D. Johnson, Y. Hu, J. Jetcheva, "A Perfomance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols", ACM/IEEE Conference on Mobile Computing and Networking, Dallas (TX), Oct. 1998.

[Est02] D. Estrin, D. Culler, K. Pister, G. Sukjatme, "Connecting the Physical World with Pervasive Networks", IEEE Pervasive Computing, 1(1):59-69, Jan. 2002.

[KubCH01] J. Kubica, A. Casal, T. Hogg, "Agent-based Control for Object Manipulation with Modular Self-Reconfigurable Robots", International Joint Conference on Artificial Intelligence, Seattle (WA), Aug. 2001, pp. 1344-1352.

[Mam02] M. Mamei, L. Leonardi, M. Mahan, F. Zambonelli, "Coordinating Mobility in a Ubiquitous Computing Scenario with Co-Fields", Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices at AAMAS 2002, Bologna (I), July 2002.

[MamZL02] M. Mamei, F. Zambonelli, L. Leonardi, "Programming Context-Aware Pervasive Computing Applications with TOTA", Technical Report No. DISMI-2002-23, University of Modena and Reggio Emilia, August 2002.

[MasCE01] C. Mascolo, L. Capra, W. Emmerich, "An XML based Middleware for Peer-to-Peer Computing", 1st IEEE International Conference of Peer-to-Peer Computing, Linkoping (S), Aug. 2001.

[PicMR01] G. P. Picco, A. L. Murphy, G. C. Roman, "LIME: a Middleware for Logical and Physical Mobility", 21st International Conference on Distributed Computing Systems, IEEE CS Press, July 2001.

[Poo00] R. Poor, "Gradient Routing in Ad Hoc Networks", http://www.media.mit.edu/pia/Research/ESP/texts/poorieeepaper.pdf.

[Rat01] S. Ratsanamy,, P. Francis, M. Handley, R. Karp, "A Scalable Content-Addressable Network", ACM SIGCOMM Conference 2001, San Diego (CA), ACM Press, Aug. 2001.

[RomJH02] G.C. Roman, C. Julien, Q. Huang, "Network Abstractions for Context-Aware Mobile Computing", 24th International Conference on Software Engineering, Orlando (FL), ACM Press, May 2002.

[RowD01] A. Rowstron, P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems", 18th IFIP/ACM Conference on Distributed Systems Platforms, Heidelberg (D), Nov. 2001.