*T*uples *O*n *T*he *A*ir: a Middleware for Context-Aware Multiagent Systems

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 activities in multi agent systems. 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 agents. On the one hand, the middleware propagates tuples across a network on the basis of application-specific patterns, to define sorts of "computational fields", and adaptively re-shape the resulting distributed structures accordingly to changes in the network scenario. On the other hand, application agents can locally "sense" these fields and can rely on them for both acquiring contextual information and carry on complex coordination activities in an adaptive way. Several examples in different scenarios show the effectiveness of such an approach.

1. Introduction

The emergence of distributed computing scenarios such as mobile, peer-to-peer, and pervasive computing, calls for novel approaches to support and facilitate the development and execution of distributed applications. In particular: *(i)* In all the above scenarios, the activities of software agents are often contextual, i.e., strictly related to the environment in which the hosting device situates (e.g., a room, a street, or a Web of computational resources), requiring agents to execute in a *context-aware* way by dynamically acquiring information about the surrounding environment. *(ii)* The intrinsic openness and dynamics of the above scenarios, in which new devices and agents can leave, arrive, and die at any time, requires applications to be *adaptive* in their activities and interactions, and capable of dealing with such dynamics in an unsupervised way. *(iii)* The adherence to the above requirements must not clash with the need of promoting *simplicity* of both applications and supporting infrastructures, also to meet the needs of resource-constrained and power-limited devices.

The above issues do not find effective solutions in traditional approaches to distributed computing. There, distributed software agents are usually assumed as fixed and connected with each other via reliable network connections. Consequently: *(i)* application agents, 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), requiring them to explicitly acquire such information via external services; *(ii)* agents 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.

The contribution of this paper is to present TOTA ("Tuples On The Air"), a novel middleware infrastructure explicitly conceived as a support for multi agent systems in dynamic network scenarios. The key objectives of TOTA, only partially achieved by similar proposals in the area are: *(i)* to promote uncoupled and adaptive interactions by locally providing application agents with simple, yet highly expressive, contextual information; and *(ii)* to actively support adaptivity by discharging application agents 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. On the one hand, 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. On the other hand, application agents have simply to locally "sense" tuples to acquire contextual information, to exchange information with each other, and to implicitly and adaptively coordinate their activities. To take a metaphor, we can imagine that

TOTA propagates tuples in the same way as the laws of nature provides propagating fields in the physical space: although particles do not interact directly with each other, driven by locally perceived force fields, exhibit globally coordinated and adaptive motion patterns.

The following of this paper is organized as follows. Section 2 motivates our work and discusses related work. Section 3 overviews the TOTA approach. Section 4 details the architecture of the TOTA middleware, its implementation, and its API. Section 5 describes some application examples, showing how can TOTA be effectively applied to different scenarios. Section 6 concludes and outlines future works.

2. Motivations and Related Works

A number of recent proposal 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. Of course, due to the vast amount of recent literature on these topics, our analysis of related works is necessarily partial and focused on those that, to our knowledge, are the closest to our proposals.

Some approaches propose relying on shared data structures as the basis for uncoupled interactions and context-awareness. For instance, the Lime [10] and the XMIDDLE [7] middlewares exploit tuple spaces and XML trees, respectively, as the basis for interaction in dynamic network scenario. Each agent 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 agents 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 contextual information is typically strictly local (limited to raw data acquired from network nodes) and is of no support to agents in acquiring a more global perspective and in achieving complex coordination patterns.

The approach proposed in [13] 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 the gas station 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 agents: once agents have gathered contextual information they are left on their own, without being supported in their coordination activities. For instance, even if agents can dynamically acquire reciprocal knowledge, they may be thereafter forced to interact in a direct way to coordinate their activities with each other.

Anthill [1] is a framework built to support design and development of adaptive peer-to-peer applications, that exploits an analogy with biological adaptive systems [2, 9]. Anthill consists of a dynamic network of peer nodes, each one provided with a local tuple space ("nest"), in which distributed mobile agents ("ants") can travel and can indirectly interact and cooperate with each other by leaving and retrieving tuples 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 agents 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.

An area in which the problem of achieving context-awareness and adaptive coordination has been effectively solved (and that, consequently, has partially influenced our proposal) is amorphous computing [8]. The particles constituting an amorphous computer have the basic capabilities of propagating sorts of abstract computational fields ("tropisms") in the network, and to sense and react to such fields. In particular, particles can transfer an activity state towards directions described by fields' gradients, so as to make coordinated patterns of activities (to be used for, e.g. self-assembly) emerge in the system independently of the specific structure of the network (which is, by definition, amorphous). Although serving totally different purposes, such an approach enables, via the single abstraction of tropisms, to both diffuse contextual information and to organize adaptive global coordination patterns.

3. Tuples on the Air: Overview

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 agents' interaction, and thus also leading to a 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 agents. 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 agents 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. The specific nature of the network scenario determines how each node can found its neighbors: e.g., in a MANET scenario, TOTA nodes are found within the range of their wireless connection; in the Internet they can be found via an expanding ring search (the same used in most Internet peer-to-peer systems).

Upon the distributed space identified by the dynamic network of TOTA nodes, each agent 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. 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 fields propagated in any sort virtual space. Such space must be supported by an appropriate routing mechanism allowing distant network nodes to be neighbors in the virtual space. For example, to realize a virtual toroid space on a network whose nodes are connected on a line, then a routing mechanism enabling one end of the line to see the other end as a neighbor would be required.

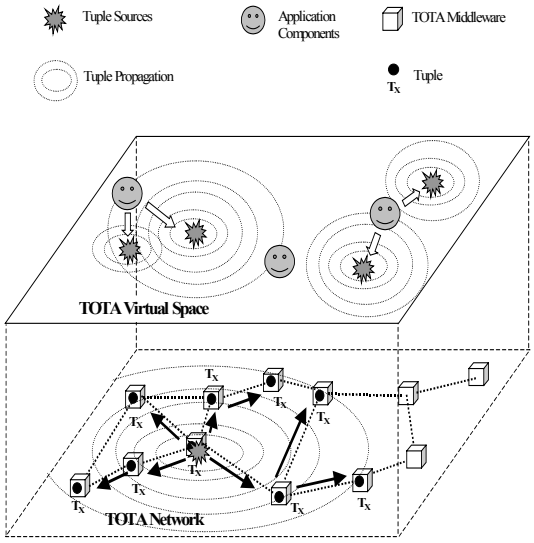

Figure 1: **The General Scenario of TOTA: application agents live in an environment in which they can inject tuples that autonomously propagate and sense tuples present in their local neighborhood. The environment is realized by means of a peer-to-peer network in which tuples propagates by means of a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

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. For instance, Figures 2, 3, and 4, show how the structure of a distributed tuple can be kept coherent by TOTA in a MANET scenario, despite dynamic network reconfigurations. From the application agents' point of view, executing and interacting basically reduces to inject tuples, perceive local tuples, and act accordingly to some application-specific policy.

The overall resulting scenario – making it sharp the analogy with the physical world anticipated in the introduction – is that of applications whose agents: *(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.

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

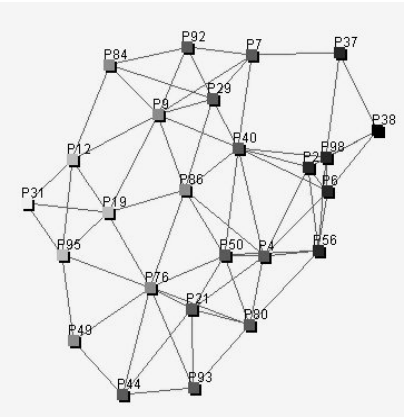

Figure 2: **P31 propagates a tuple that increases its value by one at every hop. Tuple hop-value is represented by node darkness.**

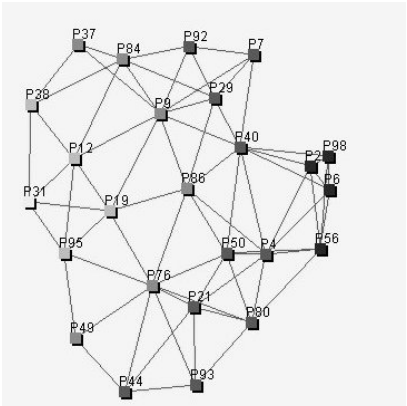

Figure 3: **P37 and P38 moves closer to the source and their tuples automatically change values to maintain the distributed tuple coherency.**

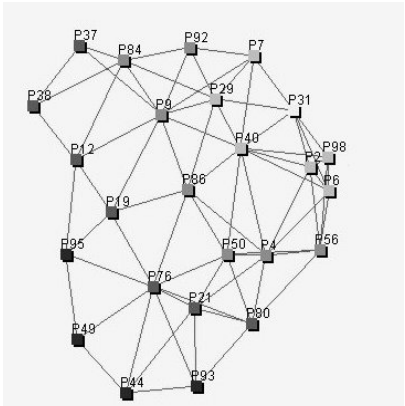

Figure 4: **When the tuple source P31 moves, all tuples are updated to take into account the new topology.**

Software agents 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, agents 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 to inject a tuple in the network and have such distributed tuple be interpreted as a query at the application-level, by having other agents in the network react to the income of such tuple, i.e., by injecting a reply tuple propagating towards the enquiring node.

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 (e.g. all the nodes within 10m, for a Bluetooth 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 whether 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 (e.g., the MAC address) together with a progressive counter for all the tuples injected by the node.

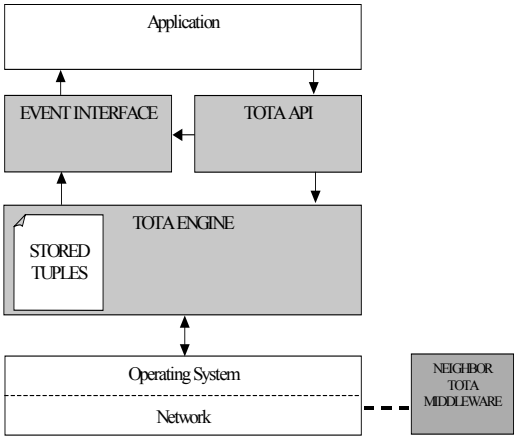

Figure 5: **TOTA Middleware**

From the architecture point of view, the TOTA middleware is constituted by three main parts (see Figure 5): *(i)* the TOTA API, described in detail in 4.3, is the main interface between an agent and the middleware. It provides functionalities to let an agent to inject new tuples in the system, to access the local tuple space, or to place subscriptions in the event interface. *(ii)* 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 from the application level, and to apply the propagation rule of received tuples and to re-propagate 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. *(iii)* 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.

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 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 three IPAQs and three laptops on which to run the system. Since the effective testing of TOTA would require a larger number of devices, we have implemented an 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.

The snap-shots of Figure 2, 3, and 4 are actually rendered via the implemented emulator (the emulator is freely accessible as a Java applet from: **http://polaris.ing.unimo.it/didattica/curriculum/marco/Web-Tota/simulator.html**).

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 agent.

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 in the agent (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 method **propagate** 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. This approach is very handy when programming new tuples, because there is no need 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 particular the four abstract methods are embedded in the overall structure of the method **propagate** in this way:

```
if(this.decidePropagation()) {

 this.doAction();
 updatedTuple = this.changeTupleContent();
 if(this.decideStore()) {
     totam.store(updatedTuple);
    }

}
```

The method (*decidePropagation*) is executed upon arrival of a tuple on a new node, and returns true if the tuple has to be propagated in that middleware, false otherwise. So for example an implementation of this method that returns always true, realize a tuple that floods the network because it is propagated in a breadth first, expanding ring way to all the nodes in the network. Vice-versa an always-false implementation creates a tuple that does not propagate, because the expanding ring propagation is blocked in all directions. In general more sophisticate implementations of this method are used to narrow down tuples spreading to avoid network flooding.

The method (*doAction*) allows the tuple to undertake some operations on the local middleware in which it is going to be propagated. Basically, through the use of this method a tuple can access the TOTA API provided by the middleware, and thus read, inject or delete tuples. This mechanism allows for example to globally delete a distributed tuple: in TOTA, there is not any notion of distributed tuple deletion, but a tuple can propagate through the network and specify in its *doAction* method to locally delete any occurrence of an intended tuple.

The method (*changeTupleContent*) creates a new tuple that updates the previous one in the propagation process. This method is used because a tuple can change its content during the propagation process. Basically before propagating a tuple to a new node, the tuple is passed to this method that returns an eventually modified version of it, to be propagated. So for example, if the tuple content is an integer value and this method returns a tuple having that value increased by one, we create a tuple whose integer content increases by one at each propagation hop.

The method (*decideStore*) returns true if the tuple has to be stored in the local middleware, false otherwise. This method can be used to create transient tuples, i.e., tuples that roam across the network like messages (or events) without being stored. Eventually, once the above process determined that a tuple has to be locally stored, a method of the middleware with restricted access (the *store* method) is invoked on the local TOTA middleware (*totam*) to store a tuple in the local tuple space. More details on the TOTA programming model, with also code examples, can be found in a companion paper [5].

5. Application examples

In this section, to prove the generality of our approach, we will show how to exploit TOTA to solve several problems typical of dynamic network scenarios, by simply implementing different tuples' propagation rules.

5.1. 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. Motion coordination has been widely studied in several research areas: robotics, simulations, pervasive computing, multi agent systems, etc. Among the others, a particularly interesting and successful approach is the one that exploit the idea of potential fields to direct the movement of the involved entities [4, 6]. In this section, we will consider the problem of letting a group of mobile agents (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 [2]: flocks of birds stay together, coordinate turns, and avoid each other, by following a very simple swarm algorithm. Their coordinated behavior can be explained by assuming that each bird tries to maintain a specified separation from the nearest birds and to match nearby birds' velocity. To implement such a coordinated behavior in TOTA, each agent can generate a tuple *T=(C,P)* with following characteristics:

C= (FLOCK, nodeName,val)

P= ("val" is initialized at 2, propagate to all the nodes decreasing by one in the first two hops, then increasing "val" by one for all the further hops)

To test the above coordination mechanism we used the emulator: the snap-shots of Figure 6-7 show a MANET scenario in which a group of four agents (in black) proceeds in a flock, maintaining a one hop distance. The other nodes in the network remain still and just store and forward flocking tuples.

5.2. 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 idea of the routing algorithm we will try to implement is the following [11]: 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

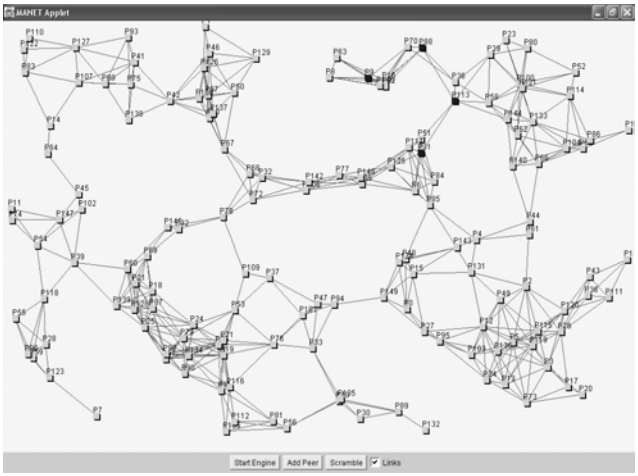

Figure 6. **Flocking emulation: step 1**

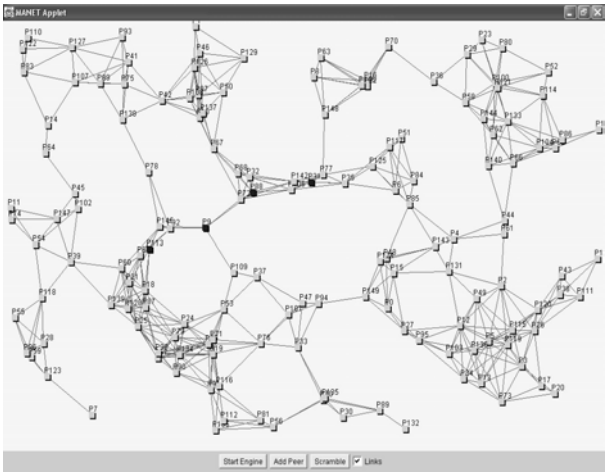

Figure 7. **Flocking emulation: step 2**

Thus creating a distributed data structure in which the *val* field assumes the minimal value at specific distance from the source (e.g., 2 hops). This distance expresses the intended spatial separation between agents in the flock. To coordinate movements, agents 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 agents maintain an almost regular grid formation by clustering in each other *val* fields' minima.

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 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 [3]. 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 such as CAN [12] and Pastry [14], to provide content-based routing in the Internet peer-to-peer scenario. In these models, peers forming an unstructured and dynamic community need to exchange data and messages not on the basis of the IP addressing scheme, but rather on the basis of the content of messages (e.g., "I need the mp3 of Hey Jude, no matter who can provide it to me"). To this end, these systems propose a communication mechanism based on a publish-subscribe model and rely on a properly built overlay space. A peer publishes information by sending them to a particular point of the overlaid space, while another read such information by looking for it in the same point of space (typically the process involves a hash function shared between all the peers, that maps keywords, associated to the information content, to points in space). TOTA can realize such systems by using a first layer of tuples defining the overlay space and then other tuples whose propagation rules let the tuples propagate efficiently in the overlaid space.

5.3. Gathering Information

Building on the previous routing mechanism, another application that can be easily realized upon TOTA is a system to enable agents 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 think at having 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. In other words:

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 [13].

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 (e.g. Lime [10] and XMIDDLE [7]). For instance, the Lime functionalities can be simply achieved by proper propagation rules, limiting the propagation of tuples to a set of direct neighbor nodes with specific characteristics.

6. Conclusions and Future Works

The TOTA middleware, by relying on distributed tuples to be propagated over a network as sorts of physical fields, provides an effective support to support distributed applications in dynamic network scenarios. As we have tried to shown via several application examples: *(i)* applications can exploit TOTA to gather contextual information and to carry on distributed coordinated activities in a simple way; *(ii)* the TOTA middleware provides to automatically reshape tuples' distribution accordingly to the dynamics of the network and of applications, thus promoting spontaneous re-organization and adaptivity.

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 models to rule accesses to distributed tuples and their updates. Indeed, the whole area of mobile and computing opens new challenging security issues. Third, specific methodology must be defined to help developers in engineering distributed applications on the basis of

TOTA distributed tuples. Finally, as it is always the case, feedbacks from real-world applications are likely to provide further directions of improvement.

Acknowledgments. Work partially supported by Nokia Research Center Boston and by MIUR Project "MUSIQUE".

References

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

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

[3] 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.

[4] 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.

[5] M. Mamei, F. Zambonelli, L. Leonardi, "Programming Context-Aware Pervasive Computing Applications with TOTA", Technical Report No. DISMI-2002-22, University of Modena and Reggio Emilia, August 2002.

[6] 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, AAMAS 2002, Bologna, Italy, July 2002.

[7] 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.

[8] R. Nagpal, "Programmable Self-Assembly Using Biologically-Inspired Multiagent Control", 1st International Conference on Autonomous Agents and Multiagent Systems, Bologna (I), ACM Press, July 2002.

[9] V. Parunak, S. Bruekner, J. Sauter, "ERIM's Approach to Fine-Grained Agents", NASA/JPL Workshop on Radical Agent Concepts, Greenbelt (MD), Jan. 2002.

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

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

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

[13] 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.

[14] 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.