Developing Adaptive and Context-aware Applications in Dynamic Networks

Marco Mamei, Franco Zambonelli
Dipartimento di Scienze e Metodi dell'Ingegneria – Università di Modena e Reggio Emilia
Via Allegri 13 – Modena – ITALY
{ mamei.marco, franco.zambonelli }@unimo.it

Abstract
Suitable programming models and associated supporting infrastructures are required to deal with large software systems dived in complex and dynamic network environments. Here, with the aid of a case study scenario, we discuss the inadequacies of current approaches in dealing with such scenarios. Then we sketch the key characteristics of TOTA (Tuples On The Air), as a novel proposal to deal with the above inadequacy TOTA relies on tuple-based information to be spatially diffused in the network and to be exploited by application agents so as to achieve context-awareness and – consequently – to effectively coordinate with each other despite the network dynamics. Related works are also discussed.

1. Introduction

Modern distributed systems are characterized by being dived in complex and dynamic network environments. First, networks increasingly characterized by the presence of ephemeral nodes (as, e.g., smart objects distributed sensors [Est02]), coming and leaving at any time in unpredictable ways. Second, computer-based systems can be mobile (e.g., wearables [ZamM02] and car computers). Then, distributed applications (i.e., their component agents) have to execute in a scenario in which the topology and the size of the underlying network is continuously changing, as it is the structure of the interaction patterns between agents.

This introduces peculiar challenging requirements in the development of distributed applications: (i) since new agents can leave and arrive at any time, and can roam across different environments, applications have to be adaptive, and capable of dealing with such changes in an adaptive and unsupervised way; (ii) the activities of the software systems are often contextual, i.e., related to the environment in which the systems execute, whose characteristics are typically a priori unknown, thus requiring to dynamically enforce context-awareness; (iii) the adherence to the above requirements must not clashes with the need of promoting a simple programming model suited for resource-constrained devices.

Current practice in distributed software development does not effectively address the above requirement: (i) application agents are typically strictly coupled in their interactions, thus making it difficult to promote and support spontaneous interoperations; (ii) agents are provided with either no contextual information at all or with only low-expressive information, difficult to be exploited for complex coordination activities; (iii) due to the above, the results is usually in an increase of both application and supporting environment complexity.

This paper (also with the help of a case study scenario) goes into details about the inadequacy of current approaches in dealing with dynamic network applications. Then, it sketches the main characteristics of the TOTA (“Tuples On The Air”) approach, promoting a simple yet effective way to develop adaptive and context-aware distributed applications. The key idea in TOTA is to have all interactions between application agents take place by “injecting” and propagating tuples in the network, and by local sensing of propagated tuples. Such tuples, whose propagation patterns are automatically adapted accordingly to the dynamics of the network, can express both information explicitly provided by other agents or a local view of some global property of the network. In this way, agent can carry on adaptive and context-aware activities and can easily achieve complex distributed coordination patterns.

2. On the Need of Novel Approaches

A simple case study scenario is introduced to show the inadequacy of traditional approaches in this context (and the consequent need of novel approaches).

2.1. A Case Study

The case study scenario deals with a big museum, and with the variety of people moving in it: tourists visiting the museum as well as security guards in charge of monitoring it. We can assume that each of these persons
is provided with a wireless-enabled computer assistant (e.g., a PDA). Also, we can assume the presence, in the museum, of a densely distributed network of computer-based devices, associated with rooms, corridors, art pieces, alarm systems, climate conditioning systems, etc. Such devices can be exploited for both the sake of monitoring and control, as well as for the sake of providing tourists and security guards with information helping them to achieve their goals. For tourists, such goals may include retrieving information about art pieces, effectively orientate themselves in the museum, and meeting with each other (in the case of organized groups); for security guards, these may include discovering anomalies in the museum and coordinate their movements and positions with each other to do their cooperative work in an efficient way. In the following, we will concentrate on two specific representative problems: (i) how tourists can gather and exploit information related to an art piece they want to see; (ii) how security guards can organize their respective position so as to properly monitor the museum.

Whatever specific application problem has to be addressed in the above scenario, it should meet the requirements identified in the introduction. (i) Adaptivity: tourists and security guards move in the museum, tourists are likely to come and go at any time, security guards can leave the museum or arrive to support the existing team, art pieces can be moved around the museum during special exhibitions or during restructuring works. Thus, the topology of the overall network can change with different dynamics and for different reasons, all of which have to be preferably faced without human intervention. (ii) Context-awareness: as the environment (i.e., the museum map and the location of art pieces) may not be known a priori (tourists can be visiting the museum for the first time and security guards may have been temporarily hired), and it is also likely to change in time (due to restructuring and exhibitions), application agents should be dynamically provided with contextual information helping their users to move in the museum and to coordinate with each other without relying on any a priori information; (iii) Simplicity: PDAs may have limited battery life, as well as limited hardware and communication resources. This may require a light supportive environment and the need for applications to achieve their goal with limited computational and communication efforts.

The above scenario exhibits characteristics that are typical of a larger class of modern scenarios. Among the others, traffic management and manufacturing control systems [MamLZ02], mobile robots and sensor networks [Est02].

2.2. Inadequacy of Traditional Approaches

Coordination models and middleware used so far in the development of distributed applications appear inadequate in supporting coordination activities in modern distributed computing scenarios, as the one in the previous subsection.

In direct communication models, a distributed application is designed by means of a group of agents that are in charge of communicating with each other in a direct and explicit way. Systems like Jini [Jini], as well as FIPA systems [BelPR01], support such a direct communication model. One problem of this approach is that agents, by having to interact directly with each other, can hardly support the openness and dynamics of the scenarios of interest: explicit and expensive discovery of communication partners typically supported by some sort of directory services, has to be enforced. Also, agents are typically placed in a “void” space: the model, per se, does not provide any contextual information, agents can only perceive and interact with (or request services to) other agents, without any higher level contextual abstraction. Thus, each agent has to become context aware by explicitly requesting contextual information to local services. In the case study scenario, tourists and security guards have to explicitly discover location of art pieces, of other tourists and of other security guards. Also, to orchestrate their movements, security guards must explicitly keep in touch with each other and agree on their respective movements via direct negotiation. These activities require notable computational and communications efforts and typically ends up with ad-hoc solutions – brittle, inflexible, and non-adaptable – for a contingent coordination problem.

Shared data-space models exploit localized data structures in order to let agents gather information and interact and coordinate with each other. These data structures can be hosted in some centralized data-space (e.g., tuple space), as in JavaSpaces [FreHA99], or they can be fully distributed over the nodes of the network, as in MARS [CabLZ02]. In these cases, agents are no longer strictly coupled in their interactions, because tuple spaces mediate interactions and promote uncoupling. Also, tuple spaces can be effectively used as repositories of local, contextual information. Still, such contextual information can only represent a strictly local description of the context that can hardly support the achievement of global coordination tasks. In the case study, one can assume that the museum provides a set of data-spaces, storing information such as nearby art pieces as well as messages left by the other agents. Tourists can easily discover what art pieces are nearby them, but to locate a farthest art piece they should query either a centralized tuple space or a multiplicity of local tuple spaces, and still they
would have to internally merge all the information to compute the best route to the target. Security guards can build an internal representation of the actual formation by storing tuples about their presence and by accessing several distributed data-spaces. However, the availability of such information does not free them from the need of negotiating with each other to orchestrate movements. In other words, despite the availability of some local contextual information, a lot of explicit communication and computational work is still required to the application agents to effectively achieve their tasks.

In event-based publish/subscribe models, a distributed application is modeled by a set of agents interacting with each other by generating events and by reacting to events of interest. Typical infrastructures rooted on this model are: Jedi [CugFD01] and Jini Distributed Events [Jini]. Without doubt, an event-based model promotes both uncoupling (all interactions occurring via asynchronous and typically anonymous events) and a stronger context-awareness: agents can be considered as embedded in an active environment able of notifying them about what is happening which can be of interest to them (as determined by selective subscription to events). In the case study example, a possible use of this approach would be to have each security guard notify its movements across the building to the rest of the group. Notified agents can then easily obtain an updated picture of the current formation in a simpler and less expensive way than required by adopting shared data spaces. However, such information still relies on agents for the negotiating the coordinated movements and does not alleviate their computational tasks (i.e., in the case study, security guards still have to explicitly negotiate their movements).

3. The TOTA Approach

The TOTA approach is mainly driven by the above considerations. It gathers concepts from both tuple space approaches [CabLZ02] and event-based ones [CugFD01] – thus preserving uncoupling in interactions – and extends them to provide agents with abstract – simple yet effective – representations of the context. The goal is to enable specific coordination activities to be implicitly and with minimal effort realized by application agents, and to be automatically adapted to the dynamics of the execution scenarios. A detailed description of the TOTA middleware can be found in [MamZL03/1], while the TOTA programming model is deepen in [MamZL03/2].

3.1. Overview of TOTA

The key idea in TOTA is to rely on distributed tuples for both representing contextual information and enabling uncoupled interactions 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 data to be transmitted between application agents but, more generally, some property of 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 find its neighbors: e.g., in a MANET scenario, TOTA peers are found within the range of their wireless connection; in the Internet they can be found via an expanding ring search. Upon the distributed space identified by the dynamic network of TOTA peers, each agent is capable of locally storing tuples and letting them diffuse through the network (see Figure 2). Tuples are injected in the system from a particular node, then they spread hop-by-hop accordingly to their propagation rule, generally being stored in the nodes they visit. Accordingly, a tuple in TOTA is defined in terms of a "content", and a "propagation rule".

The content $C$ of a tuple is basically an ordered set of typed fields representing the information carried on by the tuple. The propagation rule $P$ determines how a tuple should be distributed and propagated in the TOTA 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 do not necessarily have to be distributed replicas: by assuming different values in different nodes, tuples can be effectively used to build a distributed data structure expressing some kind of spatial/contextual information. We emphasize that 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 peers 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 peers. Similarly, when the topology changes due to peers’ movements, the distributed tuple structure automatically changes to reflect the new topology.

![Figure 1: The General Scenario of TOTA: active software agents, embedded in a distributed networked environment can inject tuples in the system that autonomously propagate or sense tuples present in their area.](image1)

![Figure 2: The TOTA Network, in which tuples propagates by means of a multi-hop mechanism.](image2)

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 policies. 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 peer.

As an additional note, the possibility of propagating tuples without storing them in the propagation nodes enables TOTA to be used not only as a distributed information repository, but also as a distributed event dispatcher [CugFD01].

### 3.2. The Case Study in TOTA

Let us consider the case study introduced in Section 2. We recall that we assume that the museum is properly instrumented with a reasonably dense number of wireless TOTA peers, e.g., associated with museum rooms and corridors as well as with art pieces, and that tourists and security guards are provided with wireless enabled PDAs running the TOTA middleware. All these devices, by connecting in ad-hoc network, define the structure of the TOTA space, which is likely to globally reflect the topology of the museum’s plan.

The first problem we face is that of enabling a tourist to discover the presence and the location of a specific art piece. TOTA makes this very simple, and let us envision two possible solutions. (i) Each art piece in the museum can propagate a tuple having as a content C its description, its location, and a value specifying the distance of the tuple from its source (i.e., of the art piece itself). In other words:

\[ C = (\text{description}, \text{location}, \text{distance}) \]

\[ P = (\text{propagate to all peers hop by hop, increasing the "distance" field by one at every hop}) \]

Then, any tourist, by simply checking its local TOTA tuple space, can discover where the art piece is located and, by following backwards the tuple up to its source, can easily reach it without having to rely on any a priori global information about the museum plan. (ii) As an alternative, we could consider that art pieces do not propagate a priori any tuple, but they can sense the income of tuples propagated by tourists – and describing the art piece they are looking for – and be programmed to react to these events by propagating backward to the requesting tourists their own location information.

The second problem we consider involves the security guards to move and monitor the museum in a coordinated way, i.e., according to a specific formation in which they have to preserve a specified distance from each other. To this end, we can take inspiration from the work done in the swarm intelligence research [Bon99]: 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.
implement such a coordinated behavior in TOTA and apply it in our case study, we can have that each security guard generates a tuple \( T = (C, P) \) with following characteristics:

\[
C = \text{(FLOCK, peerName, val)}
\]

\[
P = \left\{ \begin{array}{l}
\text{“val” is initialized at 2, propagate to all the peers decreasing by one in the first two hops, then increasing “val” by one for all the further hops)}
\end{array} \right.
\]

This tuple creates a distributed data structure in which the \( \text{val} \) field assumes the minimal value at specific distance from the source (e.g., 2 hops), distance expressing the intended spatial separation between security guards. For a tuple, the \( \text{val} \) field assumes a distribution approaching the one showed in Figure 3-left. The TOTA middleware ensures dynamic updating of this distribution to reflect peers’ movements. To coordinate movements, peers have simply to locally perceive the generated tuples, and to follow downhill the gradient of the \( \text{val} \) fields. The result is a globally coordinated movement in which peers maintain an almost regular grid formation (see Figure 3-right).

3.3. Abstractions of Space in TOTA

The type of context-awareness promoted by TOTA is strictly related to spatial-awareness. In fact, by creating an overlaid, distributed data structure, TOTA tuples intrinsically provides a notion of space in the network. For instance, 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.

Moreover, TOTA allows dealing with spatial concepts in a much more flexible way. Although at the primitive level the space is the network space, and distances are measured in terms of hops from peer to peer, it is possible to exploit more physically-grounded concepts of space. These may be required by several computing scenarios in which application agents need to interact with and acquire awareness of the physical space. For instance, if some sort of localization mechanism, whether GSP or beacon-based triangulation [HigB01], is available to peers, then tuples propagation rules can also be expressed exploiting the available spatial information. Specifically, one can bound the propagation of a tuple to a portion of physical space by having the propagation procedure – as the tuple propagates from node to node – check in the local tuple space the local spatial coordinates, so as to deciding whether to further propagate the tuple or not.

In addition, one could think at mapping the peers of a TOTA network in any sort of virtual space. This space, that must be supported by an appropriate routing mechanism allowing distant peers to be neighbors in the virtual space (e.g., the normal IP protocol), can then be used to propagate tuple so as to realize content-based routing policies, as, e.g., in CAN [Rat01].

4. Related Works

A number of recent proposals address the problem of defining supporting environments for the development of adaptive, dynamic, context-aware distributed applications.

Lime [PicMR01] exploits transiently tuple spaces as the basis for interaction in dynamic network scenario. Each mobile device, as well as each network nodes, owns a private tuple space. Upon connection with other devices or with network nodes, the privately owned tuple spaces can merge in a federated tuple space, to be used as a common data space to exchange information. TOTA subsumes and extend the Lime model. It is possible, via specific propagation rules, to have tuples distributed only in a local neighborhood, so as to achieve the same functionalities of a locally shared tuple space of Lime. In addition, propagation rules enable much more elaborated kinds of information sharing other than simple local merging of information. Similar considerations may apply with regard to other proposals for shared distributed data structures (e.g., the XMIDDLE trees [MasCE01]).

Smart Messages [Bor02], is an architecture for computation and communication in large networks of embedded systems. Communication is realized by sending “smart messages” in the network, i.e., messages which include code to be executed at each hop in the network path. The execution of the message at each hop determines the next hop in the path, making messages responsible for their own routing. Smart Messages share with TOTA, the idea of putting intelligence in the network by letting messages (or tuples) execute hop-by-hop small chunk of code to determine their propagation. However, in Smart Messages (and active messages) code is used mainly for routing or mobility purposes. In TOTA instead, the tuples are not simply routed though the network, but can be persistent and create a distributed data structure that remains stored in the network, and TOTA propagation code can also be used to change the message/tuple content, in order to realize distributed data structure and not just replicas.
An area in which the problem of achieving effective context-awareness and adaptive coordination has been addressed via a field-based approach is amorphous computing [Nag02]. 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 emerge in the system independently of the specific structure of the network. Such mechanism can be used, among other possibilities, to drive particles’ movements and let the amorphous computer self-assemble in a specific shape. A very similar approach to self-assembly has been proposed in the area of modular robots, to let a robot composed by multiple, flexibly connected components to dynamically re-shape itself to meet specific purposes [SheS02]. Although serving different purposes, both approaches share with TOTA the idea of having a single, physically inspired, mechanism to both diffuse contextual information and to organize adaptive coordination patterns.

5. Conclusions and Future Works

The TOTA approach promotes developing dynamic network applications by relying on distributed data structures, spread over a network as sorts of electromagnetic fields, and to be used by application agents both to extract contextual information and to coordinate with each other in an effective way.

Despite the fact there are a lot of examples we had been able to realize with TOTA, we still do not have a general engineering methodology or primitive tuples’ types on which to build and generalize other kind of applications. However this is not our specific limit, but it is a current general limitation: a general methodology for dealing with approaches like the one promoted by TOTA is still unknown. However, we think that such methodology could be found in the future and for sure, our final goal would be to develop a complete engineering procedure for this kind of model.

References


