Theory and Practice of Field-based Motion Coordination in Multiagent Systems

Marco Mamei, Franco Zambonelli
Dipartimento di Scienze e Metodi dell'Ingegneria
Università di Modena e Reggio Emilia
Via Allegri 13 – 42100 Reggio Emilia, Italy
email: mamei.marco@unimore.it, franco.zambonelli@unimo.it

Abstract

Enabling and ruling coordination activities between autonomous, possibly mobile, computing entities in dynamic computing scenarios challenges traditional approaches to distributed application development and software engineering. This paper specifically focuses on the problem of motion coordination, and proposes field-based coordination as a general framework to model and engineer such coordinated behaviors. The key idea in field-based coordination is to have agents' movements driven by computational force fields, generated by the agents themselves and/or by some infrastructure, and propagated across the environment. This paper shows that field-based approaches enable the definition of adaptive and effective motion coordination schemes, which can be modeled and tested by making use of a dynamical systems formalism, and which can be easily implemented either above existing middleware infrastructures or by making use of novel middleware specifically conceived for field-based coordination.

1 Introduction

Motion coordination refers to the general problem of orchestrating global patterns of movement in a set of autonomous agents situated in an environment. The term agent can refer not only to mobile software agents situated in a computational environment (e.g., a distributed world of Web resources), but more generically to any autonomous real-world entity with computing and networking capability (e.g., a user carrying on a Wi-Fi PDA, a robot, or a modern car) situated in a physical environment (e.g., a street or a building). The goals of enforcing a specific coordinated motion pattern may be various (e.g., letting agents meet together, distribute themselves in an environment according to specific spatial patterns, or simply letting them efficiently move in an environment without interfering with each other) and may have useful applications in a variety of distributed scenarios, from Internet and Grid computing, to traffic control and digital tourism.

The question on how one can effectively model and implement motion coordination patterns in an ensemble of distributed agents has to be evaluated against a number of strict requirements. First, due to both the intrinsic decentralization of the scenario and the autonomy of application components, centralized approaches are ruled out: motion coordination must be enforced in a fully distributed way. Second, due to the intrinsic dynamics and openness of most scenarios of interest, a motion coordination approach should lead to the definition of adaptive strategies, capable of working well independently of the specific characteristics of the environment and of the number and identities of the agents to be coordinated. Third, as it is always the case, an approach should not impose too much computational and communication burden on agents.

Unfortunately, traditional coordination models and middleware does not suit the needs of motion coordination in decentralized and dynamic environments. In models and systems relying on message-passing between agents [3], agents typically have to intensively communicate with each other to evaluate the current conditions of the systems (i.e., to become "context-aware") and to negotiate their current movements in the environment on the basis of an a priori encoded strategy. In models and systems relying on common interaction spaces [6; 5], agents can more easily access contextual information about the current state of the system from the shared interaction spaces, but are still left the duty of internally computing such information to implement some a priori coded strategy. The result, in both cases, is a motion coordination strategy that is achieved with high communication and computation costs, and which tends to be brittle and fragile. The alternative approach we intend to discuss in this paper is field-based coordination [8; 9; 10].

In field-based coordination, agents interact with each other and with the operational environment by simply generating and perceiving distributed data structures abstracting sorts of "gravitational fields". On the one hand, depending on the specific motion pattern to enforce, different types of fields will be generated by agents and by the environment. On the other hand, agents can locally perceive these fields and decide where to go simply on the basis of the fields they sense and of their magnitude. Eventually, a global adaptive motion pattern emerges in a fully distributed way due to the inter-related effects of agents following the fields' shape and of dynamic fields re-shaping due to agents' movements.

Here we firstly introduce the concept of field-based coordination and discuss related proposals (Section 2). Then, we show how the physical metaphor underlying field-based coordination enables modeling motion coordination problems in terms of a simply dynamical systems formalism (Section 3). Following, we discuss how the general concept of field-based coordination may be practically implemented and programmed by making use either of existing middleware infrastructure or, better, by making use of novel middleware infrastructures specifically conceived to facilitate the programming of complex motion patterns (Section 4). Eventually, some final remarks on methodological issues are reported (Section 5).

2 Field-based Approaches to Motion Coordination

Field-based coordination gets inspiration from the physical world, and in particular from the way in which masses and particles in our universe adaptively self-organize their movements accordingly to the locally perceived magnitude of gravitational/electro-magnetic fields. The key point is that gravitational fields play the very important role of locally providing agents with a global representation of the current situation of the system (i.e., of the context in which agents are situated), which is immediately usable by agents without further computation and communication activity.

2.1 Key Concepts of Field-based Coordination

Field-based approaches to motion coordination aims at providing agents with abstract - simple yet effective - representations of the context. To this end, field-based approaches delegate to a middleware infrastructure or to the agents themselves (connected in a peer-to-peer ad-hoc network) the task of constructing and automatically updating an essential distributed "view" of the system situation - possibly tailored to application-specific motion coordination problems - that "tells" agents what to do (i.e., how to move to implement a specific motion coordination pattern). Agents are simply let with the decision of whether to follow such a suggestion or not.

Operatively, the key points of a field-approach can be summarized as follows:

1. The environment is represented and abstracted by "computational fields", propagated by agents or by an environmental network infrastructure. These fields convey useful information for the agents' coordination tasks and provide agents with strong coordination-task-tailored context awareness;
2. The motion coordination policy is realized by letting agents move following the "waveform" of these fields.
3. Environmental dynamics and agents' movements induce changes in the fields' surface, composing a feedback cycle that influences agents' movement (item 2).
4. This feedback cycle let the system (agents, environment and infrastructure) auto-organize, so that the coordination task is eventually achieved in an adaptive and fully distributed way.

A computational field can be considered a sort of spatially distributed data structure characterized by a unique identifier, a location-dependent numeric value, and a propagation law determine how the numeric value should change in space. Fields are locally accessible by agents depending on their location, providing them with a local perspective of the global situation of the system. For instance, let us consider a computer-supported tourist application, where the personal agent of a museum guide can propagate, in the museum, a simple computational field whose numeric value monotonically increase from room to room as it gets farther from the source agent (See Figure 1). Clearly, such field will represent a sort of gravitational field emitted by the agent itself and will enable the personal agents of any museum visitor, from anywhere in the museum, of "sensing" where the guide is.

Clearly, the simple principle to enforce field-based motion coordination is having agents move following the local shape of specific fields. For instance, a tourist in a museum looking for the closest guide could simply instruct his personal agent to follow downhill to a local minima the overall gravitational fields emitted by the museum guides. Dynamic changes in the environment and agents' movements induce changes in the fields' surface, producing a feedback cycle that consequently influences agents' movement. For instance, should the museum guide be moving around in the museum, the field-supporting infrastructure would automatically update the corresponding computational field of the guide and would, consequently, have any tourist agent looking for a guide re-adapt his movement accordingly. This concept also makes motion coordination strategy intrinsically adaptive. For instance, should there be multiple guides in the museum, they could decide to sense each other's fields so as to stay as far as possible from each other to improve their reachability by tourists. If a guide has to go away, the same fields would allows the others to automatically and adaptively re-shape their formation.

In a given environment, the agents as well as the infrastructure can decide to propagate any kind of fields, to facilitate the achievement of application-specific problems. However, the achievement of an application-specific coordination task is rarely relying on the evaluation, as it is, of an existing computational field. Rather, in most cases, an application-specific task relies on the evaluation of an application-specific *coordination field*, as a combination of some of the locally perceived fields. The *coordination field* is a new field in itself, and it is built with the goal of encoding in its shape the agent's coordination task. Once a proper *coordination field* is computed, agents can achieve their coordination task by simply following the shape of their *coordination field*. For instance, for the guides of the museum to stay as far as possible from each other, they simply have to follow uphill a *coordination field* resulting from the sum of all the computational fields of each guide.

2.2 Related Work

Several proposals focusing on field-based approaches for motion coordination have been proposed in the last few years.

The idea of fields driving the collective movements of autonomous robots is by no means new [9]. For instance, one of the most recent re-issue of this idea, the Electric Field Approach (EFA) [8], has been exploited in the control of a team of Sony Aibo legged robots in the RoboCup

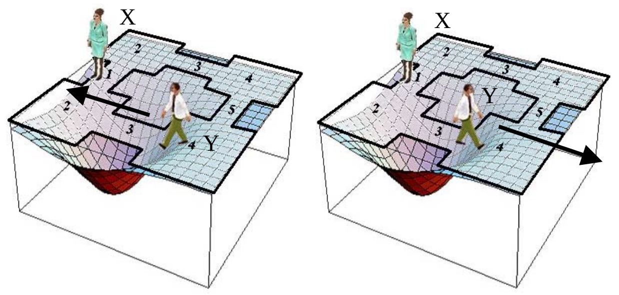

Figure 1: (left) The tourist Y senses the field generated by the tourist guide X (increasing with the distance from the source, as the numeric values in each room indicates). (left) Y agent follows the field downhill to approach X; (right) Y follows the field uphill to get farther from X. In this figure, agent Y uses the field generated by X directly as an application-specific *coordination field*.

domain. Each Aibo robot builds a field-based representation of the environment from the images captured by its head mounted cameras (stereo-vision), and decides its movements by examining the fields' gradients of this representation. The key point is that, in these approaches, it is up to agents to internally build (at high computational costs) a field-based representation of the context.

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 [1; 13]. The particles constituting an amorphous computer have the basic capabilities of propagating hop-by-hop in an ad-hoc way sorts of abstract computational fields in the network, and to sense and react to such fields. 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 [14]. A modular robot is a collection of simple autonomous actuators with few degrees of freedom connected with each other. A distributed control algorithm is executed by all actuators to let the robot assume a global coherent shape or a global coherent motion pattern (i.e. gait). Currently proposed approaches [14] adopts the biologically inspired idea of hormones to control such a robot. Hormone signals are similar to fields in that: they propagate through the network without specific destinations; their content can be modified during propagation; and they may trigger different actions for different receivers.

Shifting from physical to virtual movements, the popular videogame "The Sims" [15] exploits sorts of computational fields, called "happiness landscapes" and spread in the virtual city in which characters live, to drive the movements of non-player characters. In particular, non-player characters autonomously move in the virtual Sims city with the goal of increasing their happiness by climbing the gradients of specific computational fields (e.g., following the gradient of the fridge field when they are hungry) .

Although not specifically conceived for motion coordination, the MMASS formal model for multi-agent coordination [2] represents the environment as a multi-layered graph in which agents can spread abstract fields representing different kinds of stimuli through the nodes of this graph. The agents' behavior is then influenced by the stimuli they perceive in their location. The main application scenarios of this model are simulation of artificial societies and social phenomena, rather than motion coordination. For this reason MMASS lacks of an implemented infrastructure supporting the model, and the model per se is not operational.

3 Modeling Field-based Coordination

The physical inspiration of field-based approaches invites thinking at and modeling field-based coordinated systems in terms of dynamical systems.

3.1 The Dynamical Systems Formalism

In a continuous and unconstrained environment, fields can be represented as continuous functions of space and time, and modeling a motion coordination strategy amounts at writing the differential equations governing the motion of points (the agents) in space, driven by the gradient of a specific *coordination field* obtained as a combination of some fields of the environment. After that, integrating numerically the differential equations of the system provides a very effective way to simulate the system, and it can be regarded as an easy and powerful tool to make experiments, to quickly verify that a *coordination field* correctly expresses a coordination task, and to tune coefficients.

If we consider the agent *i*, denote its coordinates (for sake of notation simplicity we restrict to the 2-D case) as $(x_i(t), y_i(t))$ and its *coordination field* as $CF_i(x, y, t)$, the differential equations governing *i* behavior are in the form:

$$\begin{cases} \frac{dx_i}{dt} = v\frac{\partial CF_i(x,y,t)}{\partial x}(x, y, t) \\ \frac{dy_i}{dt} = v\frac{\partial CF_i(x,y,t)}{\partial y}(x, y, t) \end{cases}$$

If the motion coordination policy imply for *i* to follow uphill the *coordination field*. Changing the sign to the right member of the above equations code the agent's intention to follow downhill the *coordination field*. In fact, the gradient of the agent's *coordination field*, evaluated towards the spatial coordinates, points to the direction in which the *coordination field* increases. So the agent *i* will follow this gradient or will go in the opposite direction depending on if it wants to follow its *coordination field* uphill or downhill. We indicated with *v* a term that models the agent's speed.

The above modeling can be applied to a number of motion coordination problems, and a simple example is reported in the following of this section. However, it is worth noting that, considering those situations in which agent movements do not occur in an open and continuous space, but are constrained by some environmental conditions, the dynamical system description should be made more complex. Modeling motion in constrained environment implies the use, in equations, of either some artificial force fields that constrain agents to stay within the environment constraints (and approach that we have successfully experienced) or the adoption a domain not based on $\Re^n$, but on a more general and complex differentiable manifold.

In any case, more than analytical modeling, we think that a proper simulation environment can be effectively exploited to complement the dynamical system description and test the effectiveness of a solution in discrete and constrained environment. For instance, by simply adopting the Swarm simulation toolkit [16], one could effectively model any required spatial environment (e.g., a city street-plan or a museum map); the presence in such environment of any needed number of fields (either propagated by agents in an ad-hoc way or by a common underlying infrastructure); and the presence of any needed number of agents each with its own goals.

3.2 Modeling Examples

Let us consider the case study of having agents distribute according to specific geometrical patterns, and to let them preserve such patterns while moving. More specifically, agents can represent security guards (or security robots) in charge of cooperatively monitor a museum by distributing themselves in the museum so as to stay at specified distances from each other [7]. To implement such a coordinated behavior, we can have that each agent generates a field *(flock-field)* (the name deriving from the analogy with the behavior of flocks of birds) whose magnitude assumes the minimal value at specific distance from the source, distance expressing the desired spatial separation between security guards. To coordinate movements, peers have simply to locally perceive the generated fields, and to follow downhill the gradient of a *coordination field* obtained by considering the minimum of the perceived *flock-fields*. The expected result is a globally coordinated movement in which peers maintain an almost regular grid formation and preserve this while moving.

Analytically, the *(flock-field)* generated by an agent *i* located at (X_P^i, Y_P^i) can be simply described as follows:

$$d = \sqrt{(x - X_P^i)^2 + (y - Y_P^i)^2}$$

$$FLOCK_i(x, y, t) = d^4 - 2a^2 d^2$$

where *a* is again the distance at which agents must stay away from each other. Starting from these simple equations, one can write, for each of the agents in the set, the differential equations ruling the dynamic behavior of the system, i.e., expressing that agents follow downhill the minimum of the *(flock-field)*. For an agent i these have the form:

$$\begin{cases} \frac{dx_i}{dt} = -v \frac{\partial CF_i(x,y,t)}{\partial x}(x_i, y_i) \\ \frac{dy_i}{dt} = -v \frac{\partial CF_i(x,y,t)}{\partial y}(x_i, y_i) \end{cases}$$

$$\begin{cases} \frac{dx_i}{dt} = -v \frac{\partial min(FLOCK_1, FLOCK_2, \ldots, FLOCK_n)}{\partial x}(x_i, y_i) \\ \frac{dy_i}{dt} = -v \frac{\partial min(FLOCK_1, FLOCK_2, \ldots, FLOCK_n)}{\partial y}(x_i, y_i) \end{cases}$$

Such equations can be numerically integrated by making use of any suitable mathematical software. In our studies, we used of the Mathematica package [12]. Figure 2 shows the results obtained by integrating the above equations, for different initial conditions, and for a system composed by 4 agents. The figure shows a *x-y* plane with the trajectories of the agents of the system (i.e. the solutions of $(x_i(t), y_i(t))$ evaluated for a certain time interval) while moving in a open space. It is rather easy to see that the four agents maintain a formation, trying to maintain specified distances form each other.

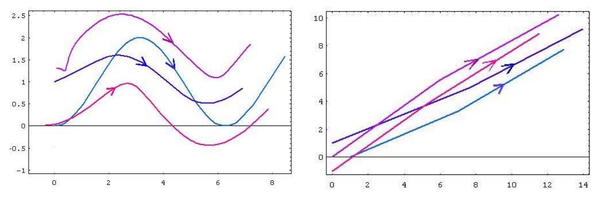

Figure 2: Solutions of the flock fields' differential equations, for different initial conditions.

4 Supporting Field-based Motion Coordination

Turning from theory to practice, the basic questions are: What software infrastructure can support a field-based model? How can it be implemented? How can it be programmed? In this section, we firstly discuss how most modern middleware can be programmed to actually support a field-based coordination model, then we show how the TOTA middleware, specifically conceived for field-based coordination, can be a much more effective solution.

4.1 Generalities on Implementing Field-based Coordination

Field-based coordination models can potentially be implemented, as an overlay service, on any middleware providing basic support for data storing, communication and event-notification. Assuming modeling fields by means of distributed data structures actually spread across the environment, what is required from the software infrastructure is to provide simple storage mechanisms (to store field values), a communication mechanism (to propagate field values to neighboring peers), and a basic event-notification and subscription mechanisms (to notify agents about changes in field values and to promptly update the distributed fields structure to support dynamic changes). For instance, in a preliminary set of experiments, field-based coordination has been implemented above a network of MARS reactive tuple spaces [5]. MARS tuples space have been allocated by IEEE 802.11 access points enabling the communication with Wi-Fi PDA, and have been programmed so as to store fields, to notify agents about local field changes and to propagate fields to neighbor access points.

The choice of implementing a field-based coordination system as an additional layer over an existing middleware, although providing for generality and portability, is not the most efficient solution. First, the mismatch between the mechanisms to be provided in the two layers tends to introduce computational and communication inefficiency. Second, the mismatch between the programming abstractions of the exploited middleware (i.e., its APIs) and those required by field-based coordination would notably complicate the implementation of motion coordination strategies.

4.2 The TOTA Middleware

TOTA (Tuples On The Air) has been explicitly developed within our research group to support field-based coordination models and likes.

In TOTA, we propose relying on distributed tuples for representing fields. Specifically, TOTA is composed by a peer-to-peer network of possibly mobile nodes, each running a local version of the TOTA middleware. 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 TOTA, fields can been realized via tuples in the form:

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.

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 (see Figure 3).

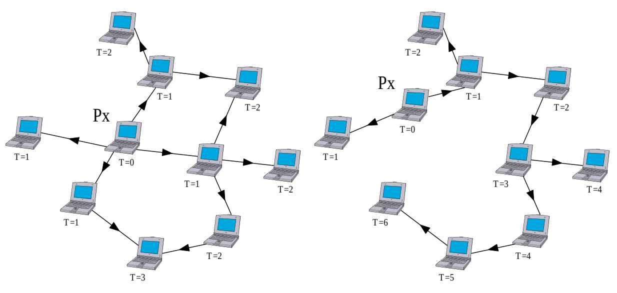

Figure 3: (left) Px propagates a tuple that increases its value by one at every hop. (right) when the tuple source Px moves, all tuples are updated to take into account the new topology

The TOTA API is the main interface to access the middleware. It provides functionalities to let the application to inject and delete tuples in the local middleware (**inject** and **delete** methods), to read tuples both from the local tuple space and from the node's one-hop neighborhood, either via pattern matching or via key-access to a tuple's unique id (**read, readOneHop, keyrd, keyrdOneHop**). Moreover, TOTA supports reactive behaviors by allowing agent to **subscribe** and **unsubscribe** to relevant events (e.g. the income of a new tuple) and to call-back agent's **react** method whenever relevant event happen. Finally, two methods (**store, move**) allow tuples to be actually stored in the local tuple space or to migrate to neighboring nodes. These methods are not part of the main API and are used only within the tuples' code.

From an implementation point of view, we developed a first prototype of TOTA running on Laptops and on Compaq IPAQs equipped with 802.11b, Familiar LINUX and J2ME-CDC (Personal Profile). 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, to implement 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.

4.3 Programming Motion Coordination in TOTA

Relying on an object oriented methodology, TOTA tuples are actually objects: the object state models the tuple content, while the tuple's propagation has been encoded by means of a specific propagate method.

When a tuple is injected in the network, it receives a reference to the local instance of the TOTA middleware, then its code is actually executed (the middleware invokes the tuple's propagate method) and if during execution it invokes the middleware move method, the tuple is actually sent to all the one-hop neighbors, where it will be executed recursively. During migration, the object state (i.e. tuple content) is properly serialized to be preserved and rebuilt upon the arrival in the new host.

Following this schema, we have defined an abstract class **TotaTuple**, that provides a general framework for tuples (i.e., for fields) programming. I.e.:

```
abstract class TotaTuple {
 protected TotaInterface tota;
 /* the state is the tuple content */
 /* this method inits the tuple, by giving a reference
 to the current TOTA middleware */
 public void init(TotaInterface tota) {
  this.tota = tota;
 }
 /* this method codes the tuple actual actions */
 public abstract void propagate();
 /* this method enables the tuple to react
 to happening events see later in the article */
 public void react(String reaction, String event)
 {
}}
```

It is worth noting that a tuple is not thread by its own, it is actually executed by the middleware, that runs the tuple's **init** and **propagate** methods. The point to understand is that when the middleware has finished the execution of the tuple's methods, the tuple (on that node) becomes a 'dead' data structure eventually stored in the middleware local tuple space.

Tuples, however, must remain active even after the middleware has run their code. This is fundamental both because agents need to sense the value of the field that a distributed tuple represents, and because their maintenance algorithm must be executed whenever the right condition appears (e.g. a new peer has been connected). To this end, tuples can place subscriptions, to the TOTA middleware as provided by the standard TOTA API. These subscriptions let the tuples remain 'alive', being able to execute upon triggering conditions.

This model for tuples gives the maximum flexibility, in that a tuple can virtually do the same things of an application installed upon the middleware. The problem is that it is too complex, and we do not foster the idea of having the programmer to write tuples code form scratch. For this reason, we have developed a tuples' class hierarchy from which the programmer can inherit to create custom tuples without worrying about most of all the intricacies of dealing with tuple propagation and maintenance.

The class **HopTuple** inherits from **TotaTuple**. This class is a template to create self-maintained distributed data structures over the network. Specifically, it implements the superclass method propagate, as shown in the following.

```
public final void propagate() {
 if(decideEnter()) {
  boolean prop = decidePropagate();
  changeTupleContent();
  this.makeSubscriptions();
  tota.store(this);
  if(prop)
   tota.move(this);
}}
```

The class **HopTuple** implements the methods: **decideEnter**, **decidePropagate**, **changeTupleContent** and **makeSubscriptions** so as to realize a breadth first, expanding ring propagation. The result is simply a tuple that floods the network increasing an hop-counter as it gets farther from the source (as in the simple example of the gravitational field of the museum guide) .

Specifically, when a tuple arrives in a node (either because it has been injected or it has been sent from a neighbor node) the middleware executes the **decideEnter** method that returns true if the tuple can enter the middleware and actually execute there, false otherwise. The standard implementation returns true if the middleware does not already contain that tuple and also if there is the tuple with an higher hop-counter. This allows to enforce the breadth-first propagation assuring that the hop-counter truly reflects the hop distance from the source. If the tuple is allowed to enter the method **decidePropagate** is run. It returns true if the tuple has to be further propagated, false otherwise. The standard implementation of this method returns always true, realizing a tuple's that floods the network being recursively propagated to all the peers. The method **changeTupleContent** change the content of the tuple. The standard implementation of this method increase an integer hop counter by one at every hop. The method **makeSubscriptions** allows the tuple to place subscriptions in the TOTA middleware. As stated before, in this way the tuple can react to events even when they happen after the tuple completes its execution. The standard implementation subscribes to network reconfiguration to implement the self-maintenance algorithm. After that, the tuple is inserted in the TOTA tuple space by executing **tota.store(this)**. Then, if the **decidePropagate** method returned true, the tuple is propagated to all the neighbors via the command **tota.move(this)**. The tuple will eventually reach neighboring nodes, where it will be executed again. It is worth noting that the tuple will arrive in the neighboring nodes with the content changed by the last run of the **changeTupleContent** method.

Programming a TOTA tuple to create a distributed data structure (i.e., a field) reduces at inheriting from the above class and overloading the four above methods to customize the tuple behavior. Here in the following, we present two examples to show the expressiveness of the introduced framework.

A **Gradient** tuple creates a tuple that floods the network in a breadth-first way and have an integer hop-counter that is incremented by one at every hop. Similarly a **FlockingTuple** creates a distributed data structure representing the *flock field* described in 3.2. In the example the tuple's value decreases in the first two hops, then increases monotonically.

```
public class Gradient extends HopTuple {
 public String name;
 public int val = 0;

 protected void changeTupleContent() {
 /* The correct hop value is maintained
 in the superclass HopBasedTuple */
 super.changeTupleContent();
 val = hop;
}}
```

```
public class FlockingTuple extends HopTuple {
 public String peerName;
 public int val = 2;
 /* always propagate */
 public HopBasedTuple changeTupleContent() {
 super.changeTupleContent();
 /* The correct hop value is maintained in
 the superclass HopBasedTuple */
 if(hop <= 2) val--;
  else if(hop >2) val++;
 return this;
}}
```

A detailed explanation of the whole tuple hierarchy is outside the scope of this paper can be found in [11].

It is rather easy now to program the agent required in the flock case study. With this regard, the algorithm followed by a **FlockingAgent** is very simple: agents have to determine the closest agent, and then move by following downhill that agent's **FlockingTuple**. In this way, agents will maintain the specified distance from each other. At start-up, each agent will propagate its **FlockingTuple**. Then it will read its local tuple space to determine the closest agent (the one whose **FlockingTuple** is lower). Then it will inspect its one-hop neighborhood to find the local shape of the **FlockingTuple** of the closest agent. Finally, it will move by following the tuple's gradient downhill. More information on this and on the problems that arise can be found in [10].

```
public class FlockingAgent extends Thread
```

```
       implements AgentInterface {
 private TotaMiddleware tota;

 public void run() {
  // inject meeting tuple to participate in meeting
  FlockingTuple ft = new FockingTuple ();
  ft.setContent(peer.toString());
  tota.inject(ft);

  while(true) {
   // read other agents' flocking tuples
   FlockingTuple query = new FlockingTuple();
   Vector v = tota.read(query);
   /* evaluate gradients and select the peer to
   which the gradient goes downhill */
   GenPoint destination = getDestination(v);
   // move downhill following the meeting tuple
   peer.move(destination);
}}}
```

5 Conclusions and Methodological Issues

In this paper we have presented the concept of field-based coordination, and have discussed both how it can be effectively modeled via dynamical systems formalism and how it can be implemented by exploiting a proper middleware.

The biggest weakness of field-based coordination is that a general methodology to help us identify, given a specific motion pattern to be enforced, which fields have to be defined, how they should be propagated, and how they should be combined in a *coordination field*, is still missing.

Independently of that, the immediate applicability of the field model is guaranteed by the possibility of getting inspiration from (and of reverse engineering) a wide variety of motion patterns to be found in nature. Phenomena such as diffusion, birds' flocking, ants' foraging, bee dances [4], to mention a few examples, can all be easily modeled with fields (i.e., in terms of agents climbing/descending a *coordination field* as the composition of some computational field), and all have practical application in mobile computing scenarios.

Nevertheless, we think that the definition of a general-purpose methodology for field-based coordination could be found in the future. In the meantime, deployment of applications and further experience will help us identifying current shortcomings of the approach and directions of improvement.

Acknowledgements Work supported by the Italian MIUR in the "Progetto Strategico IS-MANET, Infrastructures for Mobile ad-hoc Networks".

References

[1] H. Abelson, et al., "Amorphous Computing", Communications of the ACM, 43(5), May 2000.

[2] S. Bandini, S. Manzoni, C. Simone, "Heterogeneous Agents Situated in Heterogeneous Spaces", 3rd International Symposium From Agent Theories to Agent Implementations, Vienna (A), April 2002.

[3] F. Bellifemine, A. Poggi, G. Rimassa, "JADE - A FIPA2000 Compliant Agent Development Environment", 5th International Conference on Autonomous Agents (Agents 2001), Montreal (CA), May 2001.

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

[5] G. Cabri, L. Leonardi, F. Zambonelli, "Engineering Mobile Agent Applications via Context-dependent Coordination", IEEE Transaction on Software Engineering, 28(11):1040-1056, Nov. 2002.

[6] E. Freeman, S. Hupfer, K. Arnold, "JavaSpaces Principles, Patterns, and Practice", Addison-Wesley, 1999.

[7] A. Howard, M. Mataric, G. Sukhatme, "An Incremental Self-Deployment Algorithm for Mobile Sensor Networks", Autonomous Robots,special issue on Intelligent Embedded Systems, G. Sukhatme, ed., 2002, 113-126.

[8] S. Johansson, A. Saffiotti, "Using the Electric Field Approach in the RoboCup Domain", Proceedings of the Int. RoboCup Symposium. Seattle, WA, 2001.

[9] O. Khatib, "Real-time Obstacle Avoidance for Manipulators and Mobile Robots", The International Journal of Robotics Research, 5(1):90-98, 1986.

[10] M. Mamei, F. Zambonelli, L. Leonardi, "Co-Fields: A Physically Inspired Approach to Distributed Motion Coordination". IEEE Pervasive Computing, to appear 2004.

[11] M. Mamei, F. Zambonelli, "Self-mantained Distributed Tuples for Field-based Coordination in Dynamic Networks", Proceedings of the Symposium on Applied Computing, ACM Press, Cyprus (CY), March 2004.

[12] Mathematica, http://www.wolfram.com.

[13] R. Nagpal, A. Kondacs, C. Chang, "Programming Methodology for Biologically-Inspired Self-Assembling Systems", in the AAAI Spring Symposium on Computational Synthesis: From Basic Building Blocks to High Level Functionality, March 2003

[14] W. Shen, B. Salemi, P. Will, "Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots", IEEE Transactions on Robotics and Automation 18(5):1-12, Oct. 2002.

[15] The Sims, http://thesims.ea.com.

[16] The Swarm Simulation Toolkit, http://www.swarm.org.