Computing and Informatics, Vol. 34, 2015, 185–209

HOW TO DEVELOP PERVASIVE SOCIAL APPLICATIONS WITH THE SAPERE MIDDLEWARE

Gabriella Castelli, Marco Mamei
Alberto Rosi, Franco Zambonelli

Dipartimento di Scienze e Metodi dell'Ingegneria,
University of Modena and Reggio Emilia
Via Amendola 2
42122 Reggio Emilia, Italy
e-mail: {gabriella.castelli, marco.mamei, alberto.rosi,
franco.zambonelli}@unimore.it

Abstract. SAPERE ("Self-Aware Pervasive Service Ecosystems") is a general framework to support the decentralized execution of self-organizing pervasive computing services. In this paper we present the rationale underlying SAPERE and its reference conceptual architecture. Following, we sketch the middleware infrastructure of SAPERE and detail the interaction model implemented by it, based on a limited set of "eco-laws" allowing general-purpose distributed self-organizing schemes. Finally, we show how a social application can be easily implemented exploiting such an infrastructure and report on performances.

Keywords: Pervasive computing, middleware, self-organization

1 INTRODUCTION

Pervasive computing technologies are notably changing the ICT landscape, letting us envision the emergence of an integrated and dense infrastructure for the provisioning of innovative general-purpose digital services. The infrastructure will be used to ubiquitously access services for better interacting with the surrounding physical world and with social activities occurring in it.

To support the vision, a great deal of research activity in pervasive computing has been devoted to solve problems associated to the development of effective pervasive service systems including: supporting self-configuration and context-aware composition; enforcing self-adaptability and self-organization; and ensuring that service frameworks can be highly-flexible and long-lasting [22]. Unfortunately, most of the solutions so far proposed are in terms of "add-ons" to be integrated in existing frameworks [2]. The result is often an increased complexity of current frameworks and the emergence of a contrasting trade-off between different solutions.

In our opinion, there is need for tackling the problem at its foundation, conceiving a radically new way of modeling integrated pervasive services and their execution environments, such that apparently diverse issues of context-awareness, dependability, openness, flexibility, can all be uniformly addressed once, and for all, via a sound and programmable self-organization approach. This is exactly the goal of SAPERE [21], which proposes a novel nature-inspired approach to support the design and development of adaptive and self-organizing systems of pervasive computing services.

To support such a claim, in this paper, we describe an exemplary pervasive social application named "*In good company*" and illustrate how to program it according to the SAPERE abstractions. More in detail, the "*In good company*" application is set in a food court of a shopping mall scenario in which a large number of public displays provide information to users. In particular, each display aggregates the profiles of the users nearby in order to compute the average characteristics of the people around. Users can query the system to identify those places that are frequented by people sharing similar interests and profiles.

In this context, the contribution of this paper is threefold:

- we present the overall conceptual architecture of the SAPERE approach, and show how it has been realized in the SAPERE middleware
- we detail the specific approach to distributed self-organizing coordination promoted by SAPERE and discuss how that supports the effective development and execution of self-organizing pervasive applications
- we describe how to program a social application via the SAPERE framework providing commented portions of the application code and preliminary performance assessment.

The remainder of this paper is organized as follows. Section 2 introduces the SAPERE approach and sketches its reference architecture. Section 3 details how such an architecture has been realised in a middleware infrastructure for infrastructural and mobile Android devices and presents its API. Section 4 presents the self-organising approach at the base of SAPERE with the help of coding examples and Section 6 provides a preliminary performance assessment. Section 7 surveys the most relevant proposals that, in different areas, are related to the general issue of coordinating and self-organising services in a pervasive environment. Finally Section 8 offers conclusions.

2 THE SAPERE APPROACH AND ITS REFERENCE ARCHITECTURE

SAPERE takes its primary inspiration from nature, and starts from the consideration that the dynamics and decentralization of future pervasive networks will make it suitable to model the overall world of services, data, and devices as a sort of a distributed computational *ecosystem*.

As we can see from Figure 1, SAPERE conceptually architects a pervasive service environment as a non-layered *spatial substrate*, laid above the actual pervasive network infrastructure. The substrate embeds the basic interaction laws (or *eco-laws*) that rule the activities of the system, and it represents the ground on which components of different species interact and combine with each other (in respect of the eco-laws and typically based on their spatial relationships), so as to serve their own individual needs as well as the sustainability of the overall ecology. Users can access the ecology in a decentralized way to use and consume data and services, and they can also act as "prosumers" by injecting new data or service components (possibly also for the sake of controlling the ecology behavior).

For the *components* living in the ecosystem, which we generically call "agents", SAPERE adopts a common modeling and a common treatment. All agents in the ecosystem (and whether being sensors, actuators, services, users, data, or resources in general) have an associated semantic representation (in the case of pure data items, the entity and its representation will coincide), which is a basic ingredient for enabling dynamic unsupervised interactions between components. To account for the high dynamics of the scenario and for its need of continuous adaptation, SAPERE defines such annotations as living, active entities, tightly associated to the agent they describe, and capable of reflecting its current situation and context. Such *Live Semantic Annotations* (LSAs) thus act as observable interfaces of resources (similarly to service descriptions), but also as a basis for enforcing semantic and context-aware interactions (both for service aggregation/composition and for data/knowledge management).

The *eco-laws* define the basic interaction policies among the LSAs of the various agents of the ecology. In particular, the idea is to enforce on a spatial basis, and possibly relying on diffusive mechanisms, dynamic networking and composition of data and services. Data and services (as represented by their associated LSAs) will be sort of chemical reagents, and interactions and compositions will occur via chemical reactions, relying on a semantic pattern-matching between LSAs. As it is detailed later on, the set of eco-laws includes: *Bonding*, which is the basic mechanism for local interactions between components, and acts as a sort of virtual chemical bond between two LSAs (i.e., their associated agents); *Spread*, which diffuses LSAs on a spatial basis, and is necessary to support propagation of information and interactions among remote agents; *Aggregate*, which enforces a sort of catalysis among LSAs, to support distributed data aggregation; *Decay*, which mimics chemical evaporation and is necessary to garbage collected data.

Adaptivity in the SAPERE approach will not be in the capability of individual components, but rather in the overall self-organizing dynamics of the ecosystem. In particular, adaptivity will be ensured by the fact that any change in the system (as well as any change in its components or in the context of the components, as reflected by dynamic changes in their LSAs) will reflect in the firing of new eco-laws, thus possibly leading to the establishment of new bonds or aggregations, and/or in the breaking of some existing bonds between components.

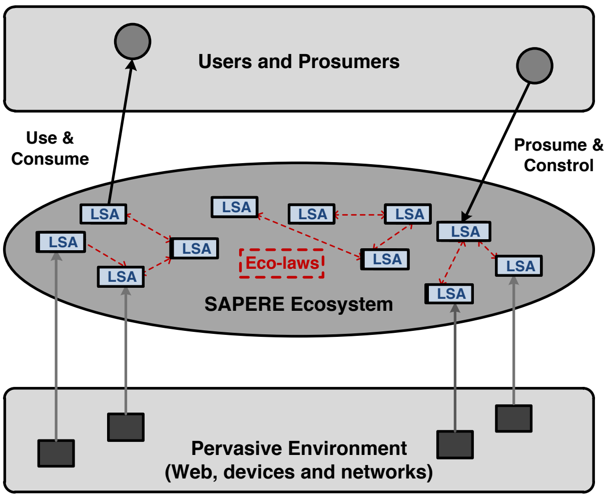


Figure 1. The SAPERE reference architecture

3 THE SAPERE MIDDLEWARE AND ITS PROGRAMMING INTERFACE

In this section we overview how SAPERE applications can be programmed by introducing the API of the SAPERE middleware and exemplifying its usage. Without having an ambition of a fully detailing the SAPERE programming approach, we intend to give readers a clue and enable them to better understand the overall SAPERE development methodology.

3.1 The Middleware

The execution of SAPERE applications is supported by a middleware infrastructure [20] which reifies the SAPERE architecture in terms of a lightweight software support, enabling a SAPERE node to be installed in tablets and smartphones. Operationally, all SAPERE nodes (whether fixed at the infrastructure level or mobile) are considered at the same level since the middleware code they run could support the same services and it provides the same set of functions.

Each SAPERE node hosts a local tuple space [8], that acts as a local repository of LSAs for local agents (LSAs are realised as tuples), and a local eco-laws engine. The LSA-space of each node is in the network with a limited set of neighbor nodes based on spatial proximity relations. Such relations consequently determine the spatial shape of the SAPERE substrate. From the viewpoint of individual agents (that will constitute the basic execution unit) the middleware provides an API to access the local LSA space, to advertise themselves (via the injection of an LSA), and to support the agents' need of continuously updating their LSAs. The update operation, that is peculiar for SAPERE in respect to traditional tuple spaces, provides support for the liveness of LSAs. In addition, such API enables agents to detect local events (as the modifications of some LSAs) or the enactment of some eco-laws on available LSAs.

Eco-laws are realized as a set of rules embedded in SAPERE node. For each node, the same set of eco-laws is applied to rule dynamics between local LSAs (in the form of bonding, aggregation, and decay) and those between non-locally-situated LSAs (via the spreading eco-law that can propagate LSAs from a node to another to support distributed interactions). From the viewpoint of the underlying network infrastructure, the middleware transparently absorbs dynamic changes at the arrival/dismissing of the supporting devices, without affecting the perception of the spatial environment by individuals.

3.2 The SAPERE API

In the SAPERE model, each agent executing on a node takes care of initializing at least one LSA (representing the agent itself), of injecting it on the local LSA space, and of keeping the values of such LSA (and of any additional LSA it decides to inject) updated to reflect its current situation. Each agent can modify only its own LSAs, and eventually read the LSAs to which it has been linked by a proper eco-law. Moreover, LSAs can be manipulated by eco-laws, as explained in the following sections.

At the middleware level, a simple API is provided to let agents inject LSA – `injectLSA(LSA myLSA)` – and to let agents atomically update some fields later – `updateLSA(field = new-value)`. In addition, it is possible for an agent to sense and handle whatever events occur on the LSAs of an agent, e.g., some match that triggers some eco-laws. For example, it is possible to handle the event represented by the LSA being bound with another LSA via the `onBond(LSA myLSA)` method.

The eco-laws assure self-adaptive and self-organizing activities in the ecosystems. Eco-laws operate on a pattern-matching schema: they are triggered by the presence of LSAs matching with each other, and manipulate such LSAs (and the fields within) according to a sort of artificial chemistry [22].

3.3 LSAs

LSAs are realized as descriptive tuples made by a number of fields in the form of "name-value" properties, and possibly organized in a hierarchical fashion: the value of a property can be a property again (called SubDescriptions in SAPERE terms). A detailed description of semantic representation of LSAs is in [18]. Here we emphasize that by building tuple-based models and extending upon them [8], the values in a LSA can be: *actual*, yet possibly dynamic and changing over time (which makes LSAs live); *formal*, not tied to any actual value unless bound to one and representing a dangling connection (typically represented with a "?").

Pattern matching between LSAs – which is at the basis of the triggering of eco-laws – happens when all properties of a description match, i.e., when for each property whose names correspond (i.e., are semantically equivalent) then the associated values match. As in classical tuple-based approaches, a formal value matches with any corresponding actual value.

For instance, the following `LSAa:(sensor-type = temperature; accuracy = 0.1; temp = 45)`, that can express the LSA of a temperature sensor, can match the following `LSAb:(sensor-type = temperature; temp = ?)`, which can express a request for acquiring the current temperature value. LSAa and LSAb matches with each other. The properties present in LSAa (e.g., accuracy) are not taken into account by the matching function because it considers only an inclusive match.

4 THE ECO-LAWS SET

Let us now detail the SAPERE eco-laws and discuss their role for a self-organization in the SAPERE ecosystem.

4.1 Bonding

Bonding is the primary form of the interaction among co-located agents in SAPERE (i.e., within the same LSA space). In particular, bonding can be used to locally discover and access information, as well to get in touch and access local services. All of which with a single and unique adaptive mechanism. Basically, the bonding eco-law realizes a sort of a virtual link between LSAs, whenever two LSAs (or some SubDescriptions within) match.

The bonding eco-law is triggered by the presence of formal values in at least one of the LSAs involved. Upon a successful pattern matching between the formal values of an LSA and actual values of another LSA, the eco-law creates the bond between the two. The link established by bonding in the presence of "?" formal fields is bi-directional and symmetric. Once a bond is established the agents holding the LSAs are notified of the new bond and can trigger actions accordingly. After bond creation, the two agents holding the LSAs can read each other LSAs. This implies that once a formal value of an LSA matches with an actual value in an LSA it is bound to, the corresponding agent can access the actual values associated with the formal ones. For instance, with reference to the `LSAa` and `LSAb` of the previous subsection, the agent having injected `LSAb`, upon bonding with `LSAa` (which the agent can detect with the `onBond` method) it can access the temperature measure by the sensor represented by `LSAb`.

As bonding is automatically triggered upon match, debonding takes place automatically whenever some changes in the actual "live" values of some LSAs make the matching conditions no longer in place.

In addition to the `?` formal field, which establishes a one-to-one bidirectional bond between component, SAPERE also makes it possible to express a "*" formal field, which leads to a one-to-many bond with multiple matchings LSAs. Moreover, the `!` formal field expresses a field that is formal unless the other `?` field has been bound. This makes it possible for an LSA to express a parametrized services, where the `?` formal field represents the parameter of the service, and the `!` field represents the answer that it is able to provide once it has been filled with the parameters.

We emphasize that the bonding eco-law mechanism can be used to enable two agents to spontaneously get in touch with each other and exchange information, all of which with a single operation and with both having injected an LSA in the space. And, in the case of the `!` field, automatically invoking a service. That is, unlike in traditional discovery of data and services [9], bonding makes possible to compose services without distinguishing between the roles of the involved agents and subsuming the traditionally separated phases of discovery and invocation.

It is worth noticing that the triggering of bonding between LSAs is driven by the content of a local LSA space. Indeed, considering a LSA with a formal field that requires bonding, three different situations may occur:

- there is only a LSA in the local tuple space that matches the formal value and a single virtual link is established between the LSAs;
- many LSAs in the local tuple space that match the formal value and a single virtual link is established between the first LSA and one randomly chosen between the matching ones;
- there are no matching LSAs, also because matching LSAs already participate in a bonding. In this case the virtual link is not established.

In [4] we discuss those issues from a theoretical point of view and propose some solutions.

4.2 Aggregate Eco-Law

The ability of aggregating information to produce high-level digests of some contextual or situational facts is a fundamental requirement for adaptive and dynamic systems. In fact, in open and dynamic environments, one cannot know *a priori* which actual information will be available (some information source may disappear, other may appear, etc.) and the availability of ways to extract a summary of all available information (without having to explicitly discover and access the individual information sources) is very important.

The aggregation eco-law is intended to aggregate LSAs together so as to compute summaries of the current system context. An agent can inject an LSA with the *aggregate* and *type* properties. The aggregate property identifies a function to base the aggregation upon. The type property identifies which LSAs to aggregate. In particular, it identifies a numerical property of LSAs to be aggregated. For example `LSAc:(aggregation_op = max; property = temp)` will trigger the aggregation eco-law that selects all the LSAs having a `temp` numerical property, computes the maximum value among them and modifies the LSAs with the result. In the current implementation, the aggregation eco-law is capable of performing most common order and duplicate insensitive (ODI) aggregation functions [14].

The aggregation eco-law supports separation of concern and allows to re-use previous aggregations. On the one hand, an agent can request an aggregation process without dealing with the actual code to perform the aggregation. On the other hand, the LSA resulting from an aggregation can be read (via a proper bond) by any other agent that needs to get the pre-computed result.

4.3 Decay Eco-Law

The decay eco-law enables that components can vanish from the SAPERE environment. The decay eco-law applies to all LSAs that specify a decay property to update the remaining time to live according to the specific decay function, or actually removing LSAs that, based on their decay property, are expired. For instance, in `LSAd: (sensor-type = temperature; temp = 10; DECAY = 1000)` it makes that LSA to be automatically deleted after a second.

The decay eco-law therefore is a kind of garbage collector capable of removing LSAs that are no longer needed in the ecosystem or no longer maintained by a component, for instance because they resulted from a propagation.

4.4 Spread Eco-Law

The above presented eco-laws basically act on a local basis, i.e., on a single LSA space. Since the SAPERE model is based on a set of networked interaction spaces, it is of course fundamental to enable non-local interactions, and specifically provide a mechanism to send information to remote LSA spaces and make it possible to distribute information and results across a network of LSA spaces.

To this end, in SAPERE we designed a so called "spread" eco-law capable of diffusing LSAs to remote spaces. One of the primary usages of the spread eco-law is to enable searches for components that are not available locally, and vice versa to enable the remote advertisement of services. For an LSA to be subjected to the spread eco-law, it has to include a `diffusion` field, whose value (along with additional parameters) defines a specific type of propagation.

Two different types of propagation are implemented in the SAPERE framework:

1. a direct propagation used to spread an LSA to a specified neighbor node, e.g., `LSAe:(...diffusion_op = direct; destination = node_x; ...)`;
2. a general diffusion capable of propagating an LSA to all neighboring SAPERE nodes, e.g., `LSAf:(...diffusion_op = general; hop = 10; ...)`,

where the `hop` value can be specified to limit the distance of propagation of the LSA from the source node.

General diffusion of an LSA via the spread eco-law to distances greater than one is a sort of broadcast that induces a large number of replicas of the same LSA to reach the same nodes multiple times from different paths. To prevent this, general diffusion is typically coupled with the aggregation eco-law so as to merge together such multiple replicas.

4.5 From Eco-laws to Distributed Self-Organization

The four above presented eco-laws form a necessary and complete set to support self-organizing nature-inspired interactions.

The four eco-laws are necessary to support decentralized adaptive behaviors for pervasive service systems. Bonding is the necessary mean to support adaptive local service interactions, subsuming necessary phases of discovery and invocation of traditional service systems. Spreading is necessary since there must be a mean to diffuse information in a distributed environment to enable distributed interactions. Aggregation and decay are necessary to support a decentralized adaptive access to information without being forced to dynamically deploy code on the nodes of the system, which may not be possible in decentralized environments.

Further, and possibly of more software engineering relevance, the eco-law set is sufficient to express a wide variety of distributed interaction schemes (or "patterns"), including self-organizing ones. Bonding and spreading can be trivially used to realize local and distributed client-server scheme of interactions as well as asynchronous models of interactions and information propagation. Coupling spreading with aggregation and decay, however, it is possible to realize also those distributed data structures necessary to support all patterns of nature-inspired adaptive and self-organizing behaviors, i.e., virtual physical fields, digital pheromones, and virtual chemical gradients [2].

In particular, aggregation applied to the multiple copies of diffused LSAs can reduce the number of redundant LSAs so as to form a distributed *gradient* structures [10], also known as *computational force fields.* As detailed in [12], many different classes of self-organized motion coordination schemes, self-assembly, and distributed navigation can be expressed in terms of gradients. For instance, Figure 2 shows how it is possible to define a `GuideAgent` that builds, with its LSA, a distributed computational field spanning at a 1 hop distance and another `SearchAgent` that follows such a field uphill by binding to a "guide" LSA available in its local tuple space and accessing the information in it.

In addition, spreading and aggregation can be used together to produce distributed self-organized aggregations, i.e., dynamically computing some distributed property of the system and have the results of such computation available at each and every node of the system, as we can see from [14]. Distributed aggregation is a basic mechanism via which to realize forms of distributed consensus and distributed task allocation and behavior differentiation. For instance, the code in Figure 3 shows

```
GuideAgent extends SapereAgent{

public void setInitialLSA(){
    addProperty("name", "guide");
    addProperty("diffusion_op", "general");
    addProperty("hop", 1);
    addProperty("aggregation_op", "min");
    addProperty("source", "GuideAgent");
    }

...
}

SearchAgent extends SapereAgent{

public void setInitialLSA(){
   addProperty("name", "guide");
   addProperty("hop", "*");
    }

 public void onBond(Event e) {
  LSA l = e.getLSA();
  float d = computeDistanceFromHop(e.getProperty("hop"));
         print("guide distance = "+d);
         print("go toward "+e.getProperty("source")); }
          }

...
}
```

Figure 2. Generating and navigating distributed data structures. The agent `Guide` uses the spread eco-law combined with aggregation to create field-like data structures, that agent `Search` can then detect and follow downhill.

how it is possible to aggregate temperature information from multiple distributed sensors. Many `SensorAgent` exist in the ecosystem and inject a LSA with the temperature value in the local LSA space. A `RequestingAgent` can inject an LSA that can adaptively compute the maximum temperature of sensors exploiting the aggregation eco-law also in remote tuple spaces combining it with spreading.

By bringing also the decay eco-law into play, and combining it with spreading and aggregation, one can realize pheromone-based data structures, which makes possible to realize a variety of bio-inspired schemes for distributed self-organization [2]. In particular, while general diffusion and progressive decay can be used to realize diffusible and evaporating pheromone-like data structures, direct propagation can be used to navigate by following pheromone gradients.

```
RequestingAgent extends SapereAgent{

public void setInitialLSA(){
    addProperty("aggregation_op", "max");
    addProperty("property", "temp");
    addProperty("diffusion", "general");
    addProperty("hop", 1);
    addProperty("source", "GuideAgent");
  }

  ...
}

SensorAgent extends SapereAgent{

  public void setInitialLSA(){
    float t = sample();
    addProperty("temp", t);
    }

  public void run() {

   while(true) {
        float t = sample();
        updateProperty("temp", t);
  }

    ...
}
```

Figure 3. Distributed aggregation. Many temperature sensors `1N` exist in the ecosystem. A `Requesting` agent can inject an LSA that, by combining spreading and aggregation, can adaptively compute the maximum temperature of sensors.

5 AN EXEMPLARY PERVASIVE SOCIAL APPLICATION

SAPERE naturally accounts for application scenarios characterized by people interacting with each other and with network displays providing services to guide, to inform, to raise attention of passing by people. Among many possible applications we propose "In good company" – a distributed application allowing people to spend some time with friendly persons or in whatever way sharing common affinities.

5.1 Application Description

"In good company" has been developed to suit with crowded and wide environments (as a museum or a shopping mall) where the occupants of each location (or room, floor, building, etc.) may vary dynamically during the day, and with them, the characterization of "how friendly" (i.e. the concentration of people sharing common interests) is that location for a person.

To distinguish from existing "find your friends" applications, the realization in SAPERE terms results as totally privacy safe, since personally identifiable information (as real name/surname, GPS positions, or checkins at determined place) is not publicly exposed or requested. Each user of the application is only requested to provide his/her ID from an adopted Social Network (e.g. Facebook, Google Plus, Foursquare, etc.) thus only the information the user has already set in his/her public profile concurs to calculate how friendly (or affine) is a restaurant of a mall food court, a showroom in a museum, a waiting room of a train station.

In this section we concentrate on making the "In good company" app for the food court in a shopping mall (despite the implementations for other application scenarios would result totally equivalent from a technical point of view) where the public display of each restaurant, beside showing information about the meals, waiting time, promotions, directions to services, etc. also interconnects to each other displays (thus restaurants) of the mall (see Figure 4). Before detailing how we coded the application in SAPERE terms, the application use case could be described as follows:

1. an user running the app (see Figure 5, on the left) on his/her mobile phone approaches the mall food court willing to launch "In good company";
2. user's request for friendly locations is brought from one display of the ecosystem and forwarded to other displays (each of them associated to a food provider of the court);
3. for each given restaurant, its display takes care of polling its customers (using the app) to provide a measure of friendship affinity towards the requesting user;
4. each display aggregates such measures and push back the answer to the requesting user;
5. given such information the user will decide in which restaurant to have lunch and which group of people eventually to join (see Figure 5, on the right).

5.2 SAPERE Implementation

This section describes the realization of the 5 steps stated above using the elements SAPERE provides to developers. For this app we assume that a SAPERE node with the app code is running both on users' smartphones, both on restaurants display stands. From the point of view of deployed agents, the application is realized through the use of 2 agents running on each phone and 4 agents running on each display of the ecosystem.

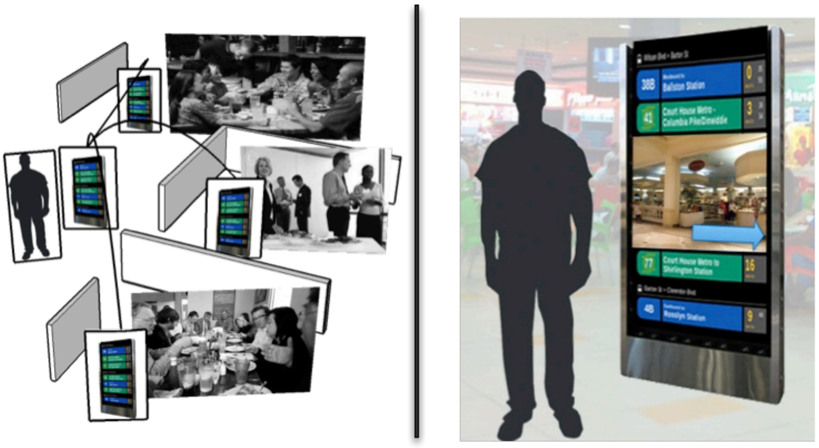


Figure 4. On the left, the idea behind the "In good company" app: a network of restaurants where public displays help people to find the right place in which to have lunch. On the right, a public display providing directions to a restaurant beside other public utility information (e.g. restaurant waiting time, directions to restrooms, availabilities of public transportations.

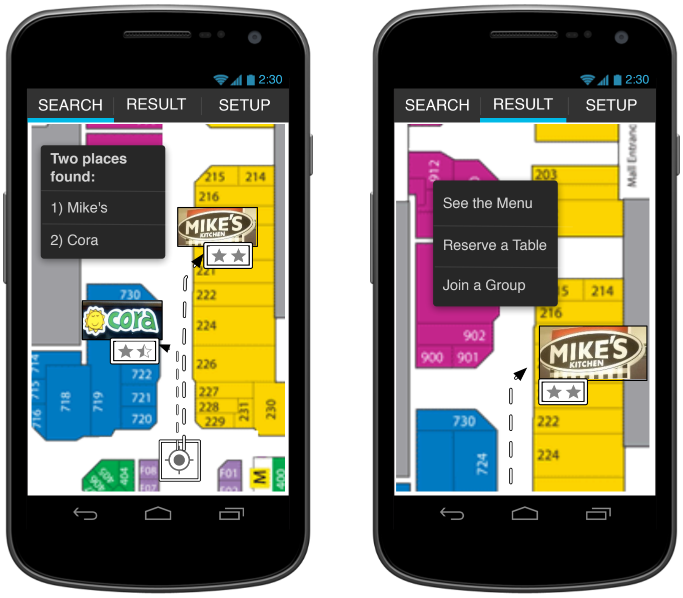


Figure 5. Two screenshots from the prototyped version of the "In good company" app. On the left, the Search menu providing a path to selected restaurant and a measure, with stars, of the affinity. On the right, the Result menu where the user, as an option, can see the menu, reserve a table or join an existing one.

The first step of the app sees the `CustomerAgent` running on the user's smartphone to detect a display of the food court (the one at the entrance or one of the others) and to inject an LSA with the user profile (in our example, linking to a Facebook user ID) and an affinity query, as shown in the code snippet in Figure 6. The agent is notified by the space of events happening to its own LSA (bonding, propagations, etc.), thus enabling it to react modifying the content of the LSA itself. In particular, the `CustomerAgent` injects a LSA in the local smartphone space to be propagated to the display_Room1 node (as declared by properties *diffusion_op* and *destination*) to start the app.

```
CustomerAgent extends SapereAgent {
  ...

  public void setInitialLSA(){
    addProperty("in-good-company-app", "query");
    addProperty("fb-id", "#123456");
    addProperty("diffusion_op", "direct");
    addProperty("destination", "display_Room1");
  }
  ...
}
```

Figure 6. The code managing the basic actions of the CustomerAgent

```
InteractionManagerAgent extends SapereAgent
{
 ...

  public void onBond(Event e) {
    Lsa l = e.getLSA(); // bonded LSA
    new QueryPropagatorAgent
              (l.getProperty("fb-id"));
  }
  ...
}
```

Figure 7. The code of the InteractionManagerAgent to match a query request

On the other hand, in each display an `InteractionManagerAgent` exposes a LSA to match a query request in order to start the app interaction. Once the two LSAs have bound, the second step of the application sees the `InteractionManagerAgent` to start a `QueryPropagatorAgent` (see code snippet in Figure 7) to propagate the affinity query (with a gradient indicating the number of hops and decay time) to sur-

```
public class QueryPropagatorAgent extends SapereAgent {
  LSA myLSA;
  String my_fbId;

  public QueryPropagatorAgent(String my_fbId){
    super();
    this.my_fbId = my_fbId;
  }

  public void setInitialLSA(){
    addProperty("diffusion_op", "general");
    addProperty("hop", "10");
    addProperty("decay", 10);
    addProperty("fb-id", my_fbId);
    addProperty("remote-affinity-request", "10");
    addProperty("source", display_name);
  }
  ...
}
```

Figure 8. The code of the QueryPropagatorAgent to propagate the affinity request into the network of displays

rounding displays to find an affine place to have lunch (see code snippet in Figure 8 and the Figure 10 for a gradient propagation example).

During the third step, other people participating to the app receive (through the displays of the ecosystem) the affinity request: once the LSA request for affinity has been propagated to remote displays through a gradient, the `QueryManagerAgent` on each display starts propagating an affinity evaluation to all the smartphone directly connected to itself as depicted in code snippets in Figure 9.

```
QueryManagerAgent extends SapereAgent {
   ...

  public void onBond(Event e) {
    addProperty("fb-id", my_fbId);
    addProperty("diffusion_op", "direct");
    addProperty("destination", "all");
  }
  ...
}
```

Figure 9. The code of the QueryManagerAgent to match a query request

Thus the `AffinityEvaluatorAgent` running on the mobile phone evaluates the affinity between its own user and the received Facebook profile.

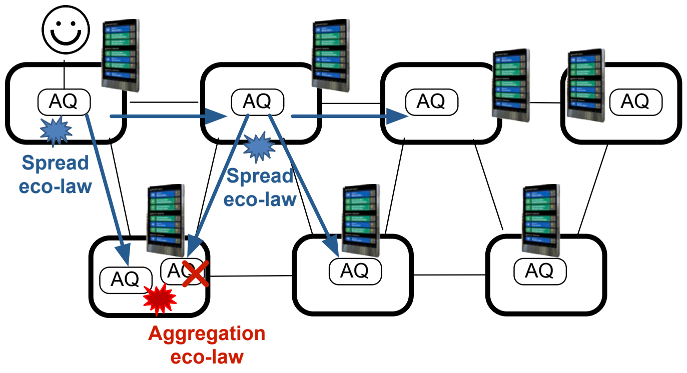


Figure 10. The restaurant Agent propagates the affinity query (AQ) – with a gradient indicating the number of hops and decay time – to surrounding displays

Affinity is calculated given a positive weight to the fact that such Facebook ID is between users' friends or, in any case, if interests in the user public page find a match.

After each display has aggregated the received affinity values, a per-restaurant aggregated affinity value is propagated back with a self-organizing pattern called chemotaxis (see Figure 12) by the `QueryAggregatorAgent` as in code snippet in Figure 11.

The `QueryPropagatorAgent` running on the display which made the initial request aggregates (using the max operator) the received values (see code snippet in Figure 13) and passes the result to the CustomerAgent on the smartphone. The user can now follow the suggestion about where having lunch "In good company" (this final step is omitted for the sake of brevity).

6 EXPERIMENTS

The purpose of this section is to demonstrate that the SAPERE middleware is ready to handle a realistic application ("In good company" app), distributing the efforts between the nodes and tolerating – linearly – an increasing topology complexity in terms of nodes of the network (e.g. the restaurants) and occupants (e.g. the people occupying them). Reported results concern the middleware cycle timing, the number of middleware cycles and the number of LSAs that each node must shoulder to run the app (tests have been performed on a 2 GHz Apple MacBook pro however similar tests have been conducted on an Android mobile device with a predictable worsening of performances reaching the 60 %). For each of the experiments we executed 1 000 runs (and averaged the results) with varying number of nodes – from 10 to 50 – and

```
QueryAggregatorAgent extends SapereAgent {
  LSA myLSA;
  String my_fbId;

  public QueryAggregatorAgent(String my_fbId){
    super();
    this.my_fbId = my_fbId;
  }

  public void setInitialLSA(){
    addProperty("aggregation_op", "avg");
    addProperty("property", "affinity-eval");
    addProperty("fb-id", my_fbId);
    addProperty("remote-affinity-request", "10");
    addProperty("chemotaxis", <spread_id_value>);
  }
  ...
}
```

Figure 11. The code of the QueryAggregatorAgent that triggers the aggregation of affinity values and propagates it back exploiting the chemotaxis

varying number of people occupying them – reaching 10, 30 or 50 units. Despite the fact that the selected configurations of nodes and people most of the time could be considered "realistic" for a mall food court, our experiments are not defying a working range for the application, but rather tracing a trend for those parameters to impact workload of nodes.

Average Cycle Timing. Figure 14 reports the average timing requested for each node to complete a middleware cycle. As we can see from the figure, the timing performances span from very few ms to nearly 60 and are mostly influenced by number of the nodes rather than by number of people populating the nodes. This is rather intuitive since the more the nodes, the more spread propagations have to be handled by each single node (and thus the more are the copies of each LSAs flowing around the network).

Average Cycles per Node. In Figure 15 we report the average number of middleware cycles that each node of the network should hold to realize the app. In the worst case, to serve 2 500 people (50 people on each of the 50 nodes) the app requests in average 16 cycles for each node to complete. Given the fact that a middleware cycle hardly can keep more that 60 ms (from first experiment) a waiting time strictly less than 1 second represents a fully tolerable delay for a person to enjoy the service.

Average LSAs per Node. Figure 16 reports the average number of LSAs populating each node of the network for the above depicted configurations. For

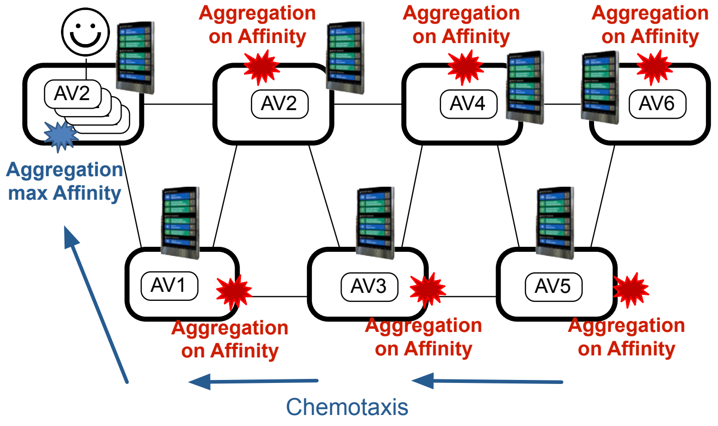


Figure 12. Per-restaurant aggregated affinity values (AVn) is propagated back with chemotaxis

```
QueryPropagatorAgent extends SapereAgent {
  ...

 public void onPropagation (Event e) {
    removeGradient();
    addProperty("aggregation_op", "max");
    addProperty("aggregation_field"),"affinity-eval");
  }
}
```

Figure 13. After the gradient is propagated, the QueryPropagatorAgent triggers the aggregation of affinity values calculated in remote nodes

a number of nodes equal to 10, 30 and 50 we also report the minimum and maximum number measured. Also in this case, in the worst configuration (50 people on each of the 50 nodes) the number of LSAs populating a node does not pass 300 units. Given the fact that a LSA hardly overcomes 0.5 KB in memory, this makes the app suitable to run on low-end personal computer or even in modern PDAs or smartphones.

7 RELATED WORKS

7.1 Coordination

The issue we face in this article can be framed as the problem of finding the proper coordination model for enabling and ruling interactions of pervasive services. We take as a ground the archetypal LINDA model, which simply provides for a black-

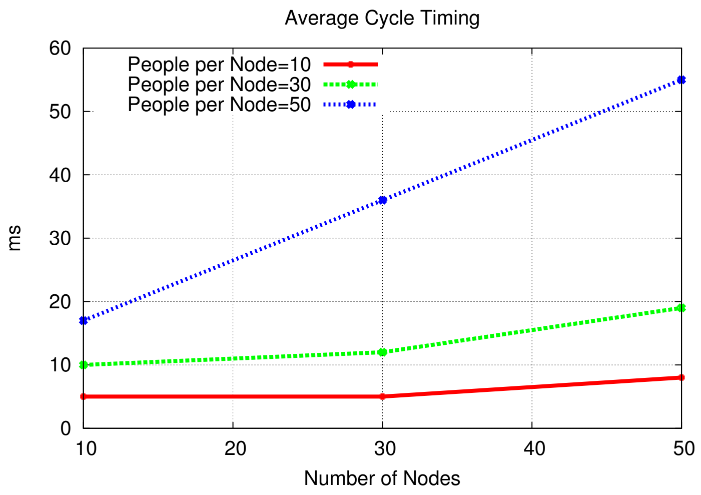


Figure 14. The average timing requested for each node to complete a middleware cycle

board with associative matching for mediating component interactions through insertion/retrieval of tuples. Then, we followed the idea of engineering the coordination space of a distributed system by some policy "inside" tuple spaces, following the pioneer works of approaches like Tucson [15] and MARS [3]. In particular, our proposal tries to extend these models to include bio-inspired ecological mechanisms, by fine-grained and well structured chemical-like reactions. In particular, the coordination approach we propose in this paper originates from the chemical tuple space model in [19], though with some notable differences:

1. here we provide a detail notational framework to flexibly express eco-laws that work on patterns of LSAs and affect their properties;
2. the chemical concentration mechanisms proposed in [19] to exactly mimic chemistry is not mandatory here – though it can be achieved by a suitable design of rate expressions;
3. the way we conceive the overall infrastructure and relationship between agents and their LSAs goes beyond the mere definition of the tuple-space model.

Our approach aims at specifically tackling coordination infrastructures for pervasive systems, which calls for dictating specific mechanisms of fuzzy matching, diffusion, context- and spatial-awareness, and agent-LSA interaction.

7.2 Situatedness and Context-Awareness

Considering the issues of situatedness and context-awareness, extensions or modifications to the traditional SOAs have been recently proposed to address adaptivity

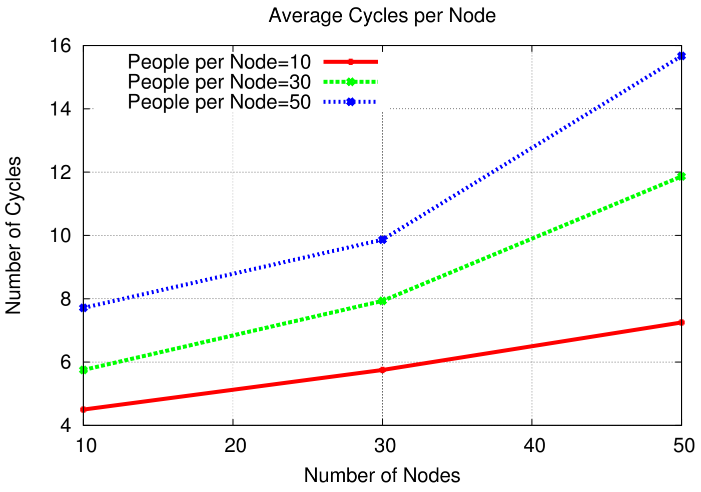


Figure 15. The average number of cycles requested for each node to handle a varying number of nodes and occupants

in pervasive environments. Similarly to our approach, in PLASTIC [1] service descriptions are coupled with dynamic annotations related to the current context and state of a service, to be used for enforcing adaptable forms of service discovery. However, our approach gets rid of traditional discovery of services and enforces dynamic and adaptive service interaction via simple chemical reactions and a minimal middleware.

In many proposals for pervasive computing environments and middleware infrastructures, the idea of "situatedness" has been promoted by the adoption of shared virtual spaces for services and components interactions. The pioneering system Gaia [17] introduces the concept of active spaces, that is active blackboard spaces acting as the means for service interactions. Later on, a number of Gaia extensions where proposed to enforce dynamic semantic pattern-matching for service composition and discovery [6] or access to contextual information [5]. Other related approaches include: Egospaces [11], LIME [13] and TOTA [12]. Our model shares the idea of conceiving components as "living" and interacting in a shared spatial substrate (of tuple spaces) where they can automatically discover and interact with one another. Yet, our aim is broader, namely, to dynamically and systemically enforce situatedness, service interaction and data management with a simple language of chemical reactions, and most importantly, enacting an ecological behavior.

7.3 Self-Organization

Several recent works exploit the lessons of adaptive self-organizing natural and social systems to enforce self-awareness, self-adaptivity and self-management features in

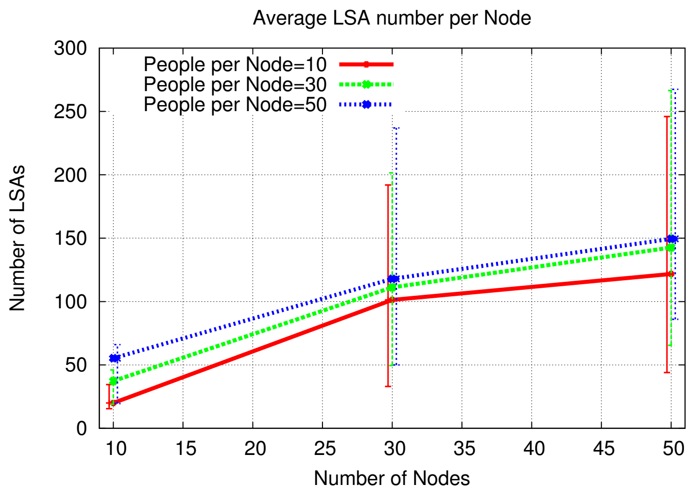


Figure 16. The average number of LSA populating each node to handle a varying number of nodes and occupants. Error bars show the min and max LSA values for 10, 30 and 50 nodes.

pervasive computing systems. At the level of individual component modeling, these proposals take the form of either situated reactive agents or proactive and goal-oriented ones [16]. At the level of interaction models, these proposals typically take the form of a specific nature- and socially-inspired interaction mechanisms [2], enforced either at the level of component modeling or via specific middleware-level mechanisms.

We believe our framework integrates and improves these works in three main directions:

1. it abstracts from the specific internal characteristics of components (no matter whether they are simple reactive components or complex goal-oriented ones) and rather proposes an approach that seamlessly applies to both cases;

2. it tries to identify an interaction model that is able to represent and subsume the diverse nature-inspired mechanisms under a unifying self-adaptive abstraction (i.e. the semantics chemical reactions);

3. the ecological approach we undertake goes beyond most of the current studies that limit to ensembles of homogeneous components, defining a suitable framework for supporting the vision of novel pervasive and internet scenarios as made up of self-adaptive devices and services, that autonomously cooperate for the creation of global services.

8 CONCLUSIONS

In this paper we introduced the SAPERE paradigm and proved that implementing complex applications using the SAPERE middleware offers multiple advantages over established approaches such as client- server models or aspect-oriented programming. In particular, SAPERE eases the development of a wide area of pervasive computing applications since it supports well the decentralized nature of the developed application, managing the system with a massive number of distributed and interacting components. The reported experiments show that the overall performance of the framework is in line and can support a large number of pervasive computing applications.

As plan for future work, we intend to experience the SAPERE approach with a number of innovative services in the area of crowd management and urban computing, by exploiting the ecosystem of pervasive displays as a technical testbed.

Acknowledgments

Work supported by the EU FET Unit, under grant No. 256873.

REFERENCES

[1] Autili, M.—Benedetto, P.—Inverardi, P.: Context-Aware Adaptive Services: The Plastic Approach. Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering (FASE '09), Springer, Berlin, Heidelberg, 2009, pp. 124–139.

[2] Babaoglu, O.—Canright, G.—Deutsch, A.—Di Caro, G.—Ducatelle, F.—Gambardella, L.—Ganguly, G.—Jelasity, M.—Montemanni, R.—Montresor, A.: Design Patterns from Biology for Distributed Computing. ACM Trans. Auton. Adapt. Syst., Vol. 1, 2006, No. 1, pp. 26–66.

[3] Cabri, G.—Leonardi, L.—Zambonelli, F.: MARS: A Programmable Coordination Architecture for Mobile Agents. IEEE Internet Computing, Vol. 4, 2000, No. 4, pp. 26–35.

[4] Castelli, G.—Mamei, M.—Rosi, A.—Zambonelli, F.: Behavior Predictability Despite Non-Determinism in the SAPERE Ecosystem. Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW 2012), Lyon, France, 2012, pp. 205–210.

[5] Costa, P. D.—Guizzardi, G.—Almeida, J. P. A.—Pires, L. F.—Van Sinderen, M.: Situations in Conceptual Modeling of Context. Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), October 16–20, 2006, Hong Kong, China, IEEE Computer Society, 2006, Workshops, p. 6.

[6] Fok, C. L.—Roman, G. C.—Lu, C.: Enhanced Coordination in Sensor Networks Through Flexible Service Provisioning. Coordination Languages and Models, Springer-Verlag, LNCS, Vol. 5521, June 2009, pp. 66–85.

[7] Omicini, A.—Zambonelli, F.: Coordination for Internet Application Development Journal Autonomous Agents and Multi-Agent Systems, Vol. 2, 1999, No. 3, pp. 251–269.

[8] Gelernter, D.: Generative Communication in Linda. ACM Trans. Program. Lang. Syst., Vol. 7, 1985, No. 1, pp. 80–112.

[9] Huhns, M. N.—Singh, M. P.: Service-Oriented Computing: Key Concepts and Principles. IEEE Internet Computing, Vol. 9, 2005, No. 1, pp. 75–81.

[10] Fernandez-Marquez, J. L.—Arcos, J. L.—Di Marzo Serugendo, G.—Viroli, M.—Montagna, S.: Description and Composition of Bio-Inspired Design Patterns: The Gradient Case. Proceedings of the 3rd Workshop on Biologically Inspired Algorithms for Distributed Systems, New York, NY, USA, 2011, pp. 25–32.

[11] Julien, C.—Roman, G. C.: Egospaces: Facilitating Rapid Development of Context-Aware Mobile Applications. IEEE Trans. Software Eng., 2006, pp. 281–298.

[12] Mamei, M.—Zambonelli, F.: Programming Pervasive and Mobile Computing Applications: The Tota Approach. ACM Trans. Software Engineering and Methodology, Vol. 18, 2009, No. 4.

[13] Murphy, A. L.—Picco, G. P.—Roman, G.-C.: Lime: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents. ACM Trans. Software Engineering and Methodology, Vol. 15, 2006, No. 3, pp. 279–328.

[14] Nath, S.—Gibbons, R. H.—Seshan, S.—Anderson, Z. R.: Synopsis Diffusion for Robust Aggregation in Sensor Networks. Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, Baltimore, MD, USA, 2004, pp. 250–262.

[15] Omicini, A.—Zambonelli, F.: Coordination for Internet Application Development. Autonomous Agents and Multi-Agent Systems, Vol. 2, 1999, No. 3, pp. 251–269.

[16] Ricci, A.—Omicini, A.—Viroli, M.—Gardelli, L.—Oliva, E.: Cognitive Stigmergy: Towards a Framework Based on Agents and Artifacts. Environments for MultiAgent Systems, Springer, LNAI, Vol. 4389, 2007, pp. 124–140.

[17] Roman, M.—Hess, C. K.—Cerqueira, R.—Ranganathan, A.—Campbell, R. H.—Nahrstedt, K.: Gaia: A Middleware Platform for Active Spaces. Mobile Computing and Communications Review, Vol. 6, 2002, No. 4, pp. 65–67.

[18] Stevenson, G.—Viroli, M.—Ye, J.—Montagna, S.—Dobson, S.: Self-Organising Semantic Resource Discovery for Pervasive Systems. 1st International Workshop on Adaptive Service Ecosystems: Natural and Socially Inspired Solutions, Lyon, France, 2012, pp. 47–52.

[19] Viroli, M.—Casadei, M.: Biochemical Tuple Spaces for Self-Organising Coordination. In: Field, J., Vasconcelos, V. T. (Eds.): Coordination Languages and Models, Springer-Verlag, LNCS, Vol. 5521, June 2009, pp. 143–162.

[20] Zambonelli, F.—Castelli, G.—Mamei, M.—Rosi, A.: Integrating Pervasive Middleware with Social Networks in SAPERE. International Conference on Selected Topics in Mobile and Wireless Networking, Shanghai, PRC, 2011, pp. 145–150.

[21] Zambonelli, F.—Castelli, G.—Mamei, M.—Rosi, A.: Programming Self-Organizing Pervasive Applications with SAPERE. Proceedings of the 7th Interna-

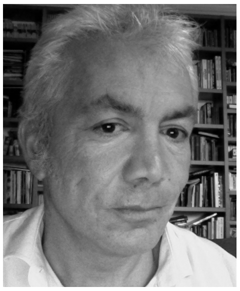
tional Symposium on Intelligent Distributed Computing, Springer-Verlag, Studies in Computational Intelligence, Vol. 511, 2014, pp. 93–102.

[22] Zambonelli, F.—Viroli, M.: A Survey on Nature-Inspired Metaphors for Pervasive Service Ecosystems. Journal of Pervasive Computing and Communications, Vol. 7, 2011, pp. 186–204.

Gabriella Castelli is a software engineer at DQuid working in the field of Internet of Things. She received her Ph.D. degree from University of Modena and Reggio Emilia in 2010. From 2010 to 2014 she was with the DISMI Department at the University of Modena and Reggio Emilia. Her research interests include pervasive computing, distributed systems and coordination middleware.

Marco Mamei is Associate Professor in computer science at the University of Modena and Reggio Emilia. He received his Ph.D. degree in computer science from the same University in 2004. He has been visiting researcher at Telecom Italia Lab (IT), Nokia Research Center (USA), Harvard University (USA), Cycorp Europe (SLO) and Yahoo Research (ES). His current research interests include mobility data analysis and applications for pervasive and mobile computing.

Alberto Rosi is an E/E Integration Manager at Ferrari Spa. He received his master degree in engineering management from the University of Modena and Reggio Emilia and his Ph.D. degree in industrial innovation from the same University. From 2006 to 2013 he was a part of the Agent Group at the DISMI Department of Reggio Emilia. His research interests included pervasive devices (e.g. RFID tags, GPS, sensor networks), services and applications, and currently sportive cars.



Franco Zambonelli is Full Professor of computer science at the University of Modena and Reggio Emilia. He received his Ph.D. degree in computer science and engineering from the University of Bologna in 1997. His research interests include pervasive computing, multi-agent systems, self-adaptive and self-organizing systems. He has published over 80 papers in peer-review journals, and has been invited speaker at many conferences and workshops. He is the co-editor-in-chief of the ACM Transactions on Autonomous and Adaptive Systems, and he is editorial board member for the Elsevier Journal of Pervasive and Mobile Computing, for the BCS Computer Journal and for the Journal of Agent-Oriented Software Engineering. He is member of the Steering Committee of the IEEE SASO Conference. He is a scientific manager of the EU FP6 Project CASCADAS and coordinator of the EU FP7 Project SAPERE. He is ACM Distinguished Member as a scientist working in the computing field, member of the Academia Europaea, and IEEE Fellow.