Engineering Pervasive Service Ecosystems: the SAPERE Approach

Gabriella Castelli, Marco Mamei, Alberto Rosi, Franco Zambonelli
Università di Modena e Reggio Emilia

Emerging pervasive computing services will typically involve a large number of devices and service components cooperating together in an open and dynamic environment. This calls for suitable models and infrastructures promoting spontaneous, situated, and self-adaptive interactions between components. SAPERE ("Self-aware Pervasive Service Ecosystems") is a general coordination framework aimed at facilitating the decentralized and situated execution of self-organizing and self-adaptive pervasive computing services. SAPERE adopts a nature-inspired approach, in which pervasive services are modeled and deployed as autonomous individuals in an ecosystem of other services and devices, all of which interacting in accord to a limited set of coordination laws, or "eco-laws". In this paper, we present the overall rationale underlying SAPERE and its reference architecture. Following, we introduce the eco-laws-based coordination model, and show how it can be used to express and easily enforce general-purpose self-organizing coordination patterns. The middleware infrastructure supporting the SAPERE model is presented and evaluated, and the overall advantages of SAPERE are discussed in the context of exemplary use cases.

Categories and Subject Descriptors: D.2.7 [**Software Engineering**]: Distribution and Maintenance; H.4.0 [**Information Systems Applications**]: General; I.2.11 [**Artificial Intelligence**]: Multiagent Systems; C.2.4 [**Computer-Communication Systems**]: Distributed Systems

Additional Key Words and Phrases: Pervasive Computing; Middleware; Self-organization; Coordination

1. INTRODUCTION

Advances in ubiquitous, mobile and embedded computing technologies are leading to the emergence of an integrated and dense infrastructure for the provisioning of innovative general-purpose applications and services [Zambonelli 2012; Harnie et al. 2014]. The infrastructure will be used to ubiquitously access services for better interacting with the surrounding physical world and with the social activities in it [Lukowicz et al. 2012]. It is also expected that users will be able to deploy customized services and to enrich existing ones by making available their own devices and components [Campbell et al. 2008].

We are already facing the release of early pervasive services trying to exploit the possibilities opened by these new scenarios in the form of, e.g., environmental displays capable of reacting to users' presence [Alt et al. 2012; Elhart et al. 2013], car navigation systems with real-time traffic information [Riener and Ferscha 2013], and location-based social services [Pejovic and Musolesi 2013; Schuster et al. 2013]. However, the full exploitation of the emerging pervasive computing infrastructure requires innovative solutions to support the development of advanced services and applications, and in particular of services capable of flexibly and adaptively interacting with each each other on a spatial and context-aware basis.

A number of research proposals exist for middleware architectures and coordination models supporting the engineering of pervasive applications [Raychoudhury

et al. 2013]. Among the others, some recent proposals absorb concepts from self-organizing and self-adaptive natural systems [Omicini 2012; Zambonelli and Viroli 2011]. Getting inspiration from nature can in fact be effective to support spontaneous composition of services, to support spatiality and situatedness, and to promote self-organization and self-adaptation. Unfortunately, many nature-inspired solutions are proposed in terms of "add-ons" to be integrated in existing frameworks [Babaoglu et al. 2006]. The result is often an increased complexity and the emergence of contrasting trade-offs between different solutions.

Against this background, the SAPERE approach (*www.sapere-project.eu*) defines a fully-fledged nature-inspired and self-organizing framework for the engineering of distributed pervasive services, via which to uniformly tackle the emerging requirements of pervasive service systems.

The main contribution of this paper is to present the SAPERE architecture and coordination mechanisms. In particular, it shows that the SAPERE architecture can support the development of pervasive applications by generalizing and incorporating a number of useful nature-inspired self-organizing mechanisms (e.g., chemical bonds [Fernandez et al. 2014], stigmergy [Parunak 1997], and fields-based approaches [Mamei and Zambonelli 2009]). By using the SAPERE framework instead of relying on a number of different tools and solutions, developers can program the self-organizing and self-adapting mechanisms that are useful to their applications (or mobile apps) within the same conceptual model and architecture.

The remainder of this article is organized as follows:

—It motivates the suitability of nature-inspired approaches for the engineering of pervasive service systems, and the need for a novel synthesis of nature-inspired coordination mechanisms (Section 2);

—It introduces the reference conceptual architecture of SAPERE, its operational counterpart, and overviews the key aspects of its coordination model (Section 3);

—It details – also with the help of practical examples – the SAPERE coordination model and the associated programming model (Section 4), and discusses how it can be exploited to support adaptive self-organizing patterns (Section 5);

—It presents the design and implementation of the SAPERE middleware architecture (Section 6), and experimentally evaluates its effectiveness in supporting SAPERE applications (Section 7);

Finally, Section 8 discusses related work in the area and Section 9 concludes and sketches future development.

2. REQUIREMENTS AND MOTIVATIONS

From the analysis of the literature in this area, we identified some key requirements that are important to manage the complexity of future pervasive service scenarios. As discussed below, these requirements can hardly be met by traditional approaches to distributed systems engineering. The extent to which research proposals related to SAPERE meet these requirements is discussed in Section 8.

2.1 Requirements

Spontaneous and open interactions. Components of pervasive applications need to interact seamlessly with each other in a spontaneous way [Raychoudhury et al. 2013]. They should be provided with flexible means for discovering other components on-the-fly, engaging them in effective interaction patterns, without much a priori information or statically encoded strategies. This is also aimed at opening the scenarios to the dynamic deployment (also by users) of new components/devices/services in a fully decentralized way.

Context and Situation Awareness. Components of pervasive applications have to be supported in acquiring information about the surrounding context, and dynamically adapting to it, which calls for mechanisms to obtain an high level representation/summary of relevant contextual parameters [Ye et al. 2012; Roggen et al. 2013]. This is not only about flexibly interacting with context-data sources (as in the first requirement), but also about having mechanisms to aggregate, summarize and extract knowledge from the data – knowledge that can be possibly re-used by various application components.

Proxemic Interactions and Location-based Activities. Proxemic interactions and location based applications are at the basis of mainstream pervasive applications [Greenberg et al. 2011]. In many applications, the behavior of applications and services strongly depends on the spatial context as well as on the relative spatial positioning of users and devices. Accordingly, components should be provided with mechanisms to deal with distances and spatial information [Beal et al. 2012], and should be supported in navigating such a space.

Self-adaptation and self-organization. Pervasive computing scenarios are inherently open and decentralized (devices and service components belong to multiple stakeholders), and dynamic (due to the presence of mobile and ephemeral devices and components). This makes it impossible for humans to intervene in the system for low-level configuration, management and maintenance activities [Kephart and Chess 2003; de Lemos et al. 2013]. Rather, applications and services should be able to self-adapt (which includes self-configure, self-manage, and self-heal) their activities and self-organize their interaction patterns with little or no configuration and management efforts.

2.2 From Top-Down to Bottom-up Nature-inspired Approaches

The need for software systems to become more open, capable of dealing with the dynamics of unpredictable environments, and able to self-adapt their behavior in response to contextual changes, has been widely recognized in the areas of software engineering [Cheng et al. 2009; de Lemos et al. 2013] and distributed systems [Brazier et al. 2009; Babaoglu et al. 2006; Zambonelli and Viroli 2011].

In the area of software engineering, many proposal considers to integrate self-adaptation capabilities with software systems by coupling them with autonomic control loops [Kephart and Chess 2003]. Such control loops can dynamically monitor the behavior of the system and trigger adaptation actions on need [Kephart and Chess 2003; Vromant et al. 2011]. In the case of distributed systems this requires multiple control loops, each devoted to control a portion the system, coordinating with each other [Weyns et al. 2012].

Approaches based on the engineering of control loops achieve situation-awareness and self-adaptation by design, i.e., in a "top down" way. This may work well for systems of limited size and where the developer has control over the system as a whole. For large and decentralized systems whose components belong to different stakeholders (as pervasive services systems), solutions that are able to meet the requirements and promote context-aware and self-adaptive behavior without centralized pre-defined control strategies, i.e., in a "bottom up" way, are to be preferred [Cheng et al. 2009]. In particular, getting inspiration from natural systems and from their capability of promoting the bottom up emergence of self-organized patterns of coordinated behaviors may represent a suitable approach.

Indeed, a number of natural systems (e.g., ant colonies), put in act coordination mechanisms and express behaviors that let them meet the identified requirements spontaneously and very efficiently [Omicini 2012]. Ants, for example, interact indirectly with each other, by depositing and locally smelling pheromones in the environment. This kind of communication based on "signs" left in the environment and acting as a sort of externalized memory is generally called *Stigmergy* [Babaoglu et al. 2006; Parunak 1997]. In general, stigmergic interactions decouple interacting agents, and let interactions take place *spontaneously* (e.g., ants interact independently of their explicit will) and in an open way (ants do not need to be aware of each other). Locality in depositing and smelling pheromones enforce *proxemic and location-based* interactions. Also, pheromones inherently express some fact/event/information occurred in that portion of the environment, i.e., they promote simple forms of situation-aware interactions. Finally, the overall activities of ants in depositing pheromones and thus building complex distributed pheromones structure, and in reacting to the presence and shape of such structures, globally make globally coordinated finalized behaviors emerge in the colony, supporting *self-adaptation and self-organization.*

In addition to stigmergy, other natural systems exploit different coordination mechanisms and have inspired coordination models that are capable – with different extents – of meeting some the requirements of pervasive service systems.

Gossip-based/Epidemic Protocols are inspired by the form of gossip seen in social networks, and by the way a virus spreads in a biological community [Jelasity et al. 2005; Bicocchi et al. 2012]. These approaches well address the requirement on *context-awareness* and in particular on *situation-awareness*, in that they allow to flexibly combine and aggregate contextual information from multiple sources in a decentralized and robust way. In addition, the way to distributed and manipulate information in gossip-based/epidemic protocols is also very *open* and *adaptive.*

Fields are inspired by physical force fields and chemical gradients, aim at conveying a sense of space in distributed systems in the form of spanning-tree data structures diffused across the network to provide distance information from the source [Mamei and Zambonelli 2009; Beal et al. 2012]. These approaches well address on *proxemic interactions and location-based activities* and of a specific form of situation-awareness, i.e., *spatial awareness* enabling to effectively engineer distributed motion coordination and location-based coordination activities [Mamei and Zambonelli 2009].

Artificial Chemistries and Artificial Immune Systems, inspired by chemical reac-

tions [Fernandez et al. 2014] and immune systems [Read et al. 2012], are mechanisms based on simple-yet-flexible matching rules that use a digital description or "footprint" of the involved components to trigger interactions and compositions among them. Such approaches effectively support spontaneous and self-adaptive interactions depending on contextual conditions and in the presence of large classes of diverse components (as it can be the case for pervasive systems). However, these mechanisms typically lack the notions of space and of distributed data manipulation that exist in the other approaches.

2.3 Towards a Unified Nature-Inspired Architecture

The main idea at the basis of our work is to identify a *unified architecture and approach* to incorporate the above mechanisms and nature-inspired schemes into a coherent modeling framework, so as to get the best from each of them.

To this end, and without committing to a specific natural metaphor, one can generally consider a pervasive computing scenaas a sort of *natural ecosystems*. There, interactions and the overall coordination among components are not ruled by predefined orchestrated patterns, but are simply subjected to a limited set of *coordination laws*, acting as a sort of synthetic "laws of nature" for the ecosystem (or, shortly, "eco-laws"). From the enactment of the eco-laws, even complex patterns of interactions dynamically emerge via self-organization, the same as it happens in natural systems that evolve and organize by simply obeying natural laws (whether physical, chemical, or biological laws).

To get the best from the introduced nature-inspired mechanisms, the overall modeling of the ecosystem components and of its coordination laws should *(i)* have the flexibility of chemical systems in supporting spontaneous interactions and composition among diverse components; *(ii)* tolerate stigmergic means of interactions, i.e., rely on the components' ability to externalize contextual information to be locally shared with other components; *(iii)* support spatial abstractions means to propagate spatial information in the forms of fields, so as to support spatial awareness and spatial coordination schemes; and *(iv)* support distributed aggregation and manipulation of information, towards advanced forms of situation awareness.

As it will become clear in the course of the following sections, this is exactly the rationale behind SAPERE, its conceptual architecture, and its coordination model.

3. THE SAPERE APPROACH: OVERVIEW

3.1 Reference Conceptual Architecture

In line with a nature-inspired vision, SAPERE models and architects a pervasive service environment as a sort of abstract computational ecosystem, built around a *spatial substrate* – i.e., a set of components modeling the space where application agents execute – laid above the actual pervasive network infrastructure (Figure 1).

Interactions take place by publishing and accessing information and events (in the form of tuples called *"Live Semantic Annotations"* or "*LSAs*") in specific locations of the spatial substrate. The substrate stores such LSAs and rules how they can be manipulated (e.g., linked with each other in a sort of virtual information chemistry), how they can be perceived (as if LSAs were sorts of stigmergic signs in the environment), how they can be possibly propagated (to support the con-

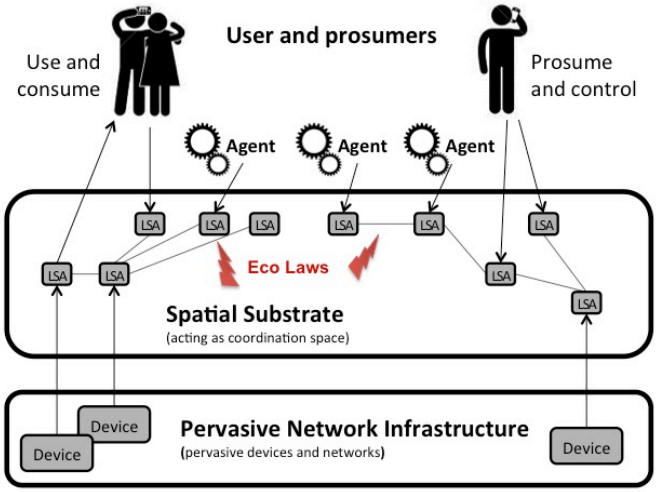


Fig. 1. The SAPERE reference architecture.

struction of distributed data structures such as fields) or aggregated (to support epidemic aggregation). Such rules – which we call "eco-laws" defines the laws ruling interactions among SAPERE individuals. In a way somewhat similar to tuple-based coordination models [Gelernter 1985; Eugster et al. 2003], we can say that SAPERE defines a *nature-inspired coordination model*, where the SAPERE spatial substrate acts as a coordination space embedding and enforcing the coordination laws (that is, the"*eco-laws*") to rule the interactions between the individuals of the ecosystem.

The population of SAPERE individuals is represented by *software agents*, each associated to one of the components that can play some roles in the provisioning of pervasive services. These include agents in charge of representing and interfacing with all pervasive devices around (e.g., ambient sensors, smart phones and individual sensors within, interactive displays and generic actuators) and agents in charge of providing specific services and algorithmic functionalities. All these agents are expected to locally access the shared substrate of the ecosystem and thus indirectly interact with each other – in respect of the eco-laws – so as to serve their own individual needs and the sustainability of the overall ecosystem.

Users themselves (e.g., via an app on a smart phone) can access the SAPERE ecosystem in a decentralized way, to use and consume data and services (as resulting from the activities of the ecosystem). Users can also act as "prosumers" by making available (in the form of SAPERE agents) new service components, new devices, or new sensor data, possibly also for the sake of controlling the ecosystem behavior.

Each SAPERE agent can take part in the ecosystem by publishing (or "injecting") in the spatial substrate a semantic description of itself in the form of an LSA tuple. An LSA is an observable interface of the agent that can encapsulate information to describe the agent itself and the data it can produce, and that can reify events occurring within the agent. To account for the high dynamics of the scenario and for its need of continuous adaptation, LSAs are "alive" in the sense that can have continuously updating content, to reflect the current situation and context of

the component they describe. For instance: an ambient sensor can be represented by an LSA describing the nature of the sensors and including the alive data that represents the currently sensed information; a pervasive display can be represented by an LSA describing the available display features and the updated information about the currently displayed information.

3.2 The Coordination Model in a Nutshell

The SAPERE coordination model considers that agents interact indirectly via LSAs, and based on how eco-laws act on such LSAs. The idea is to enforce, on a spatial and situation-aware basis and possibly relying on diffusive mechanisms, spontaneous networking and composition of data and services (as represented by the LSAs of the associated agents), capable of promoting adaptation to situations and facilitating the engineering of self-organizing coordination schemes.

The set of eco-laws that rules the interactions among SAPERE agents includes:

—*Bond*, the basic mechanism for local interactions between agents which links together spatially co-located LSAs whenever the matching conditions exist. When their LSAs are linked together, agents can access each other's information and functionalities. This is the basic mechanism enabling interactions in SAPERE, which subsumes virtual chemical composition (by linking together LSAs and, thus, the corresponding agents) and stigmergy (since LSAs are information left in the environment by some agents and perceivable by other agents).

—*Spread*, which diffuses LSAs on a spatial basis in the spatial substrate, and is necessary to support spatial-awareness by agents, propagation of information, and interactions among remote agents. In particular, spreading of LSAs is a basic mechanism via which to support advanced forms of distributed self-organization [Mamei and Zambonelli 2009].

—*Aggregate*, which can spontaneously produce new LSAs deriving from the aggregation of existing ones, and can support both the local computing of aggregated information as well as – in synergy with the spread eco-law – forms of gossip-based distributed data aggregation [Nath et al. 2004] and the creation of fields [Beal et al. 2012], both necessary to support physically-inspired self-organized coordination patterns;

—*Decay*, which mimics a sort of chemical evaporation and is necessary to garbage collect data, as well as to ensure evolvability of services via disappearing of service descriptions. In addition, along with spread and aggregate, decay makes it possible to realize pheromone-like data structures [Holldobler and Wilson 2009] and thus stigmergic coordination patterns [Babaoglu et al. 2006].

3.3 Operational Architecture

The SAPERE middleware (as represented in Figure 2 and more deeply detailed in Section 6) implements the spatial substrate in terms of a set of data spaces ("LSA spaces" – to be practically realized via tuple space technologies [Eugster et al. 2003]) distributed on the nodes that form the actual network infrastructure. Both small mobile devices (e.g., smart phones) and fixed infrastructural nodes (e.g., pervasive interactive displays) can host an LSA space, devoted to store the LSAs of local

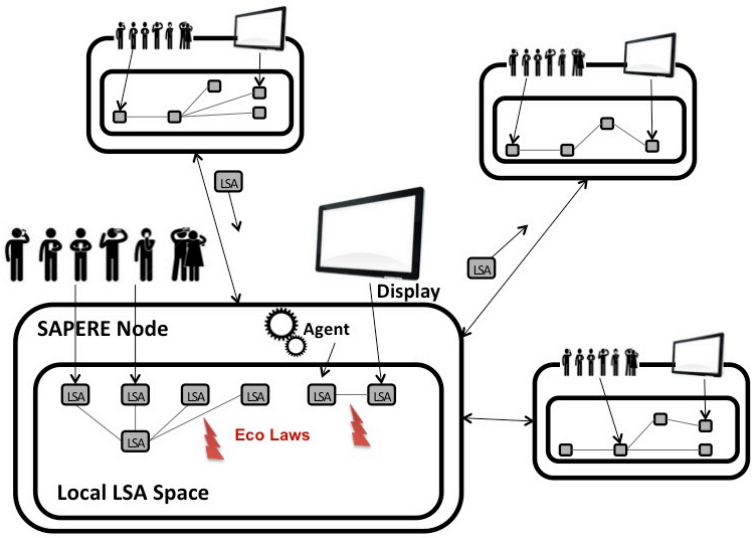


Fig. 2. The SAPERE operational architecture.

agents. LSA spaces dynamically network with each other according to their spatial relations, and on the basis of spatial strategies that can be configured within the middleware. Such strategies affect how LSAs flow and propagate from one space to neighbor ones.

On each node, specific processes execute on the LSA space to analyze the current status of LSAs and to trigger eco-laws, in the form of a set of rules that determines how and when to bond, aggregate, decay, or diffuse to which neighbor nodes the LSAs of that node.

From the agents' viewpoint, whenever an agent approaches a node or is created on it, an LSA it is automatically injected into the LSA-space of that node, making the component part of that space and of its local coordination dynamics. When a component moves away from a node, its LSA is eventually removed from that space, and possibly re-connected to one of the neighbor LSA spaces (if the associated component has moved accordingly). Agents can exploit a specific API in order to update their own LSAs, inject additional LSAs, and subscribe to local events such as the modification of some LSAs or the enactment of some eco-laws.

4. THE SAPERE COORDINATION MODEL

This section details the nature-inspired coordination model dictating how LSAs are shaped and how eco-laws act on them. Code examples will be presented to clarify the concept expressed and to show how to actually program SAPERE applications.

In particular, we focus on an exemplary pervasive computing application scenario: *information and guidance services in a smart museum*. In this scenario, users provided with smartphones and tablets visit a museum equipped with a network and sensor infrastructure. Mobile apps running on the smartphones interact with the museum infrastructure and possibly with other users to acquire information

and to provide navigation, guidance, entertainment and coordination services. The system is assumed to be able to localize users within the museum [Gu et al. 2009].

4.1 LSA - Live Semantic Annotations

Any component that takes part in a SAPERE ecosystem has to be represented by at least one LSA to be injected in the local LSA space at creation time. However, any agent can inject multiple LSAs during its lifetime. Such LSAs are linked to the corresponding agent and can reflect in real time some of its internal variables, enabling the agent to take part of the dynamics of the ecosystem. All the agents in our museum scenario will be represented via 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. The detailed description of LSA's semantic representation is not in the scope of this paper and can be found in [Stevenson et al. 2012].

By building over tuple-based models and extending upon them [Gelernter 1985], an LSA value can be *actual*, yet possibly dynamic and changing over time (which makes LSAs living entities) or *formal*, not tied to any actual value unless bound to one and representing a dangling connection (represented by the symbol "`?`").

Concerning the dynamic fields of LSAs, the idea behind the programming model is that each agent, other than taking care of injecting any needed LSAs, will also take care of keeping the values of such LSAs updated (possibly by spawning internal threads to this purpose). An agent can access and modify only the fields of its own LSAs, but it can also read the LSAs to which has been linked by an eco-law. Moreover LSAs can be accessed and modified by eco-laws, as explained in the following sections.

Pattern matching between LSAs is the mechanism at the basis of eco-laws triggering. A match between two LSAs takes place when the actual values of all the properties with the same name (or, more in general, of those properties whose concepts match according to some ontology [Stevenson et al. 2012]) correspond, or when an actual value corresponds to a formal value. More specifically, two values of a property match depending on whether they are formal or actual as from the matrix in Table I (the concept of potential value is explained a bit later).

We emphasize that, unlike in traditional tuple-space coordination models [Gelernter 1985], SAPERE does not distinguish between tuples and anti-tuples (or templates), and a unique `injectLSA` API method subsumes the read/write operations of that models. This behavior is provided by the specific way of operating of the bond eco-law, as described in the following subsection.

For example, to model temperature sensors embedded in the smart museum scenario, we can consider the following LSA: `(sensor- type = temperature; accuracy = 0.1; temp = 45)`. Such an LSA can match the following one: `(sensor- type = temperature; temp = ?)`, expressing a request to acquire information about the current local temperature. For instance, such request LSA could come from a service in need to estimate the comfort level in the museum's rooms. We emphasize that further properties presented in the first LSA (e.g., accuracy) are not taken into account by the matching function, which considers only inclusive matches.

Table I. Pattern matching rules in SAPERE.

	Actual	**Formal**	**Potential**
Actual	Y	Y	N
Formal	Y	N	Y
Potential	N	Y	N

4.2 SAPERE Programming

Programming a SAPERE application consists in developing a set of agents interacting with each other, and possibly with other agents existing in the ecosystem to meet specific the application goals. Agents are typically distributed among user devices (e.g., packing them in a mobile app) and among other pervasive computing elements deployed in the environment. In SAPERE, we enforce a notable separation of concerns between an application's computation and interaction/coordination. Computation (i.e., the main application business logic) is coded in the SAPERE agents using standard software engineering methodologies. Interaction and coordination consists in writing agents' LSAs, and let the eco-laws managing their evolution over time, and possibly promoting spontaneous service interactions and composition.

Two agents interact by reading each other's LSAs. In particular, agents express in their LSA the fact that they wish to bind with other LSAs. On the basis of the pattern matching mechanism described in Section 4.1, eco-laws will bind two matching LSAs. The execution of eco-laws triggers callback functions in the agent code, so that the application business logic can be realized.

Figure 3 shows an exemplary SAPERE agent. `TempSensorAgent` is an agent associated with a temperature sensor in the museum. The agent publishes the current temperature value via its LSA. In addition, it tries to connect to another sensor providing information about the level of CO_2 in the room to estimate a comfort parameter for the visitors. To support the programmer we developed a `SapereAgent` class to be sub-classed to create actual application agents. This class masks all the interactions with the SAPERE middleware and provides simple methods to create and update an LSA (`setInitialLSA()`, `updateLSA()`), and callback methods to be overwritten appropriately (e.g., the `onBond()` method that is called when eco-laws create new bonds, which is one of many callback methods of the API to handle the events occurring in the LSA space as induced by the eco-laws). The method `setInitialLSA()` of the API is called by the super constructor and sets the values of the LSA (lines 3-9). It is worth emphasizing that the agent expresses the need to connect to a CO_2 sensor by adding a formal "?" value associated with the `co2` property (line 9). The `run()` method (lines 16-21) is also called at the end of the super constructor and executes the main agent code. In this example the agent periodically reads the temperature sensor and updates the LSA accordingly. The `onBond()` method (lines 25-31) is called once an eco-law connects this agent to a CO_2 sensor via the pattern matching mechanism described in Section 4.1. Once

```
1. public class TempSensorAgent extends SapereAgent {
2.
3.  // the agent simply has to initialize the initial LSA
4.  // called 'myLSA' with the desired fields
5.  public void setInitialLSA (){
6.     myLSA.addProperty("sensor-type", "temperature");
7.     myLSA.addProperty("accuracy", 0.1);
8.     myLSA.addProperty("temp", readTemp());
9.     myLSA.addProperty("co2", "?");
10. }
11. // then the constructor of the SapereAgent, after having connected with the local LSA space,
12. // will automatically inject there such LSA as follows:
13. //  lsas = bindLSASpace();
14. //  injectLSA(new myLSA());
15.
16. run() {
17.  while(true) {
18.     //  this cycle reads a new value and updates myLSA
19.     sleep();
20.     String ts = Float.toString(readTemp());
21.     updateLSA("temp", ts);
22.  }
23. }
24.
25. public void onBond(Event e) {
26.  double co2 = e.getLSA().getProperty("co2");
27.  // the Event object contains a copy of the bound LSA
28.  // from which one can access the internal information
29.  double comfort = compute(readTemp(),co2);
30.  print("current comfort =  "+ comfort);
31.  myLSA.addProperty("comfort", comfort);
32.  }
33. }

34. public class CO2SensorAgent extends SapereAgent  {
35.
36.  public void setInitialLSA (){
37.   myLSA.addProperty("sensor-type", "co2");
38.   myLSA.addProperty("co2", readCO2());
39.  }
40.  run() {
41.   while(true) {
42.    sleep();
43.    updateLSA("co2", readCO2());
44.   }
45.  }
46. }
```

Fig. 3. An exemplary `TemperatureSensor` agent that publishes the current temperature value via its LSA, and connects to a CO_2 sensor to compute a comfort parameter for the visitors. An exemplary `CO2Sensor` agent reading CO_2 information and publishing it via LSA.

the bond is established interaction can proceed: the agent reads CO_2 information from the other LSA and computes the comfort parameter. Such information can be printed in a user interface and published in the LSA for others to use (lines 31-32).

For completeness, we present also (lines 34-46) the code of the `CO2SensorAgent` reading CO_2 information via the `readCO2()` method and updating its LSA accordingly. This LSA will match with the LSA of the `TempSensorAgent` to enable it reading information about the CO_2 level.

4.3 The Eco-laws Based Coordination

The eco-laws trigger reactions in the ecosystem once matches among LSAs occur. In particular, eco-laws operate on the pattern-matching schema described in Section 4.1. They are triggered by the presence of LSAs matching with each other and manipulate such LSAs (i.e., the fields within) according to a set of coordination rules [Zambonelli and Viroli 2011] (see below).

4.3.1 *Bond Eco-law.* The bond eco-law realizes a link between LSAs, whenever two LSAs (or some sub-descriptions within) match. This is the primary form of interaction among agents in SAPERE within the same LSA space. In particular, it can be exploited to locally discover and access information, as well as to get in touch and access local services.

The bond 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. In the example in Figure 3, this happens between the LSA `(sensor- type = temperature; accuracy = 0.1; temp = 45; co2=?)` of the `TemperatureSensor` agent, and the LSA `(sensor- type = co2; co2=10)` of the `CO2Sensor` agent. The link established by binding in the presence of the "`?`" formal field is bi-directional and symmetric. Once a bond is established, the agents holding the LSAs are notified of the new bond, can trigger actions accordingly and read each other's LSAs. This implies that once a formal value of an LSA matches with an actual value in another LSA it is bound to, the corresponding agent can access the actual values associated with the formal ones. If more LSAs match with a given formal value, then one match is randomly selected. Bond disruption takes place automatically whenever some changes in the actual values of some LSAs make the matching conditions no longer valid.

SAPERE also makes it possible to express a "`*`" formal field, which leads to a one-to-many bonds with multiple matchings LSAs. This is used to read *all* the LSAs matching a given signature.

Moreover, the `!` formal field – which we call "potential" formal field – expresses a field that is formal unless the other `?` field has been bound. This makes possible for an LSA to express a parameterized service, where the `?` formal field represents the parameters of the service, and the `!` field represents the answers that it is able to provide once it has been filled with the parameters.

For example the `TempSensorAgent` described before could have an LSA in the form: `(co2 = ?; comfort = !)` expressing that if it gets information about the CO_2 level - as input, it can provide a comfort value output. Another agent could have an LSA in the form `(co2 = 4; comfort = ?)`. This would trigger a reaction that automatically would complete all the formal fields in the two LSAs. With this regard, we emphasize that the bond eco-law can be used to enable two agents to discover each other, and exchange information with a single operation. Moreover, in the case of the `!` field, it also allows to automatically invoke a service. That is, unlike in traditional discovery of data and services, it allows to compose services without distinguishing between the roles of the involved agents, and subsuming the traditionally separated phases of discovery and invocation.

4.3.2 *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 sources may disappear, others may appear later, etc.), and mechanisms to extract a summary of all available information without having to explicitly discover and access the individual information sources are very important. To this end, an agent can inject an LSA with two specific `aggregate` and `type` properties.

The aggregate eco-law is triggered by those properties. It selects all the LSAs in the local space having a (numerical) property equals to the property `type`. Then it computes an aggregated value of those numerical properties on the basis of the `aggregate` value that identifies a function to base the aggregation upon (e.g., maximum, average, etc.). For example the LSA: `(aggregation-op = max; property = temp)` triggers the aggregated eco-law to compute the maximum of the `temp` values. If the in the LSA space, there are the LSAs `(temp = 10)` and `(temp = 20)`, the aggregate eco-law would produce an LSA `(type = aggregate; aggregation-op = max; temp = 20)`.

In the current implementation, the aggregate eco-law is capable of performing most common order and duplicate insensitive (ODI) aggregation functions [Nath et al. 2004; Jelasity et al. 2005], i.e., those functions whose results are independent from the order by which data is aggregated and from the possible multiple accounting of the same data items.

The aggregate eco law supports separation of concerns 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. Even more importantly, aggregation can work in combination with the spread eco-law (see later) to trigger aggregation in a fully distributed and decentralized environment, and without having to deploy specific aggregator agents in remote nodes.

4.3.3 *Decay Eco-law.* The decay eco-law enables the vanishing of components and information from the SAPERE environment. The decay eco-law applies to all LSAs that specify a `decay` property. This property expresses the remaining time to live of the LSA. Once the time to live expires, the LSA is automatically removed from the space. For instance the following LSA: `(sensor-type = temperature; temp = 10; decay=1000)` makes the LSA to be automatically deleted after 1000 time units.

The decay eco-law 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. On the other hand, for components that maintain a LSA, it is always possible to access the decay property and eventually increment its value in order to prevent the removal of the LSA. Similarly to what happens for aggregation, the decay eco-law enables distributed garbage collection of distributed LSAs without having to deploy in the various nodes of the system agents specifically devoted to that. This ensures the sustainability of the overall ecosystem. In addition, the decay function is necessary to support the realization in SAPERE of many nature-inspired

interaction patterns that rely on the spatial environment to actively play a role in making information (e.g., pheromones) to evaporate or be reinforced.

4.3.4 *Spread Eco-law.* The above presented eco-laws basically act on a local basis, i.e., on a single LSA space. However, the SAPERE model is distributed, and in particular it is grounded on interactions across a set of LSA spaces networked with each other according to some strategies (e.g., spatial proximity). Accordingly, eco-laws should also take care of ruling spatial distribution of LSAs and, accordingly, non-local interactions.

The "spread" eco-law, aims at enabling the diffusion of LSAs from one LSA space to neighbor ones, according to the concept of neighborhood defined by the topology of connections of LSAs spaces. The most basic usage of the spread eco-law is to search for components that are not available locally, or *vice versa*, to enable the remote advertisement of services. However, it is also a fundamental mechanism to support the creation of dynamic distributed data structures in support of self-organization.

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 the specific type of propagation. Two different types of propagation are implemented in SAPERE:

—`direct propagation`, a unicast that propagates an LSA to a specified neighbor node, e.g., `(...diffusion_op=direct; destination=node_x; ...)`;

—`spatial diffusion`, a multicast that propagates an LSA to all neighbor SAPERE nodes, e.g., `(...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 spatial diffusion of an LSA via the spread eco-law to distances greater than 1 is a sort of point-to-point broadcast that clearly induces a large number of replicas of the same LSA to reach the same node, multiple times, from different paths. Indeed the general diffusion, to prevent this, is typically coupled with the aggregation eco-law so as to merge together multiple copies of the same LSA that arrive on a node from different paths.

It is also worth noting that application components can use the direct propagation primitive to implement any kind of local communication scheme. For example, gossip-based information diffusion [Jelasity et al. 2005] can be easily implemented on top of direct propagation by letting an agent to communicate directly with a random subset of neighbor nodes.

5. FROM ECO-LAWS TO SELF-ORGANIZATION PATTERNS

The defined eco-laws form a limited yet highly expressive set via which is it possible to realize a large variety of nature-inspired, self-organizing and self-adaptive, coordination patterns.

5.1 Artificial Chemistries and Immune Systems

SAPERE allows to naturally model and express *artificial chemistries' and immune systems'* models. The chemical/immune-system population can be directly represented by LSA, whereas the bond eco-law can effectively model chemical reactions

and gene-antigene interactions [Fernandez et al. 2014; Read et al. 2012]. In addition, as discussed in the following, the combinations of eco-laws allow to model also other self-organizing mechanisms highlighted in Section 2.1.

5.2 Fields

Aggregation applied to multiple copies of diffused LSAs can reduce the number of redundant LSAs so as to form a distributed *field* structure [Mamei and Zambonelli 2009]. A field data structure is distributed across the nodes of the network, each of these nodes containing a copy of the LSA storing the hop-distance from the node of the network that created and injected it (see Fig. 4).

In general, field data-structures are a useful tool to encode and spread information in a distributed system. The main point of using them is that they effectively provide adaptive spatial awareness to agents. Fields in fact naturally provide a measure of distance in the network (by means of hop count from the source) and of the direction from where the information comes from (by considering the slope of the hop counts). Such kind of information is very useful in a number of pervasive applications that are closely coupled with the location of the agents.

For example, an agent monitoring a room of the museum that is fresh and not crowded could propagate a field data structure allowing other agents to sense that room and get information about how far it is located (see Figure 4).

The spread eco-law propagates the LSA hop by hop across the network. In particular, the keyword *general* specifies to recursively propagate the LSA by n hops, *aggregation_op* specifies how to aggregate the copies of the LSA. Spreading also maintains *hop_count* to indicate the number of hops from the source, *previous* to indicate from which node the field LSA comes from and *spread_id* to identify a specific field. The aggregation eco-law guarantees that redundant LSA copies are discarded and the field is properly laid out. SAPERE allows to realize this pattern effectively by moving all the burden to the eco-laws. For example the following LSA: `(diffusion-op = general; hop = 10; aggregation-op = min)` would be spread a field data-structure like the one in Fig. 4 by 10 hops.

Another agent can query for fields propagated by the room and being notified with information regarding the presence, the distance to the room, and, by using the *previous* data of the field, the approximate direction where the room is located.

It is important to notice that since the field is constructed on the basis of "ever-running" eco-laws, its global "shape" - the values of the hop counts across the network - is periodically refreshed to accommodate network churn, agents movements, and changing configuration of the entities involved. This fact makes the field and the agent's spatial awareness adaptive to changing conditions.

5.3 Stigmergy and Pheromone-based Data structures

Fields can be the basis to construct pheromone-like data structures driving agent activities [Babaoglu et al. 2006]. These data structures, mimicking pheromone trails in natural systems like ant colonies, are deployed in a distributed environment by mobile agents, and provide local information on how to explore the distributed environment. For example, a user agent finding some interesting information can start spreading a pheromone trail to allow other agents to easily reach the source (e.g., art piece) of that information, by following the trail.

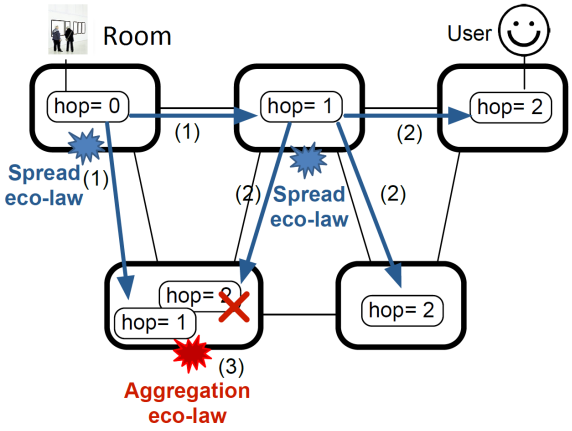


Fig. 4. Generating and navigating distributed data structures. The `Room` agent uses the spread eco-law (1) to propagate its LSA to neighbor nodes which, at their time, re-propagate a copy (2) to other nodes. Once multiple copies of the same LSA reach the same node, the aggregation eco-law preserves the copy with minimum hop number (3). This process creates field-like data structures that the `User` agent can read to get information about the presence and approximate location of the room.

Pheromones can be realized via LSAs locally deposited by agents as they move across the network, accordingly, such LSAs would be deployed on the agents path. Pheromone LSAs would be also associated with the decay eco-law to emulate pheromone evaporation. We do not report code example of this pattern in that it simply results by adding a decay property to the LSAs described for creating fields and by notably limiting the hop radius at which the field can propagate – or by removing propagation entirely – to obtain a pheromone trail that is stored only in the nodes actually visited by the agent.

5.4 Distributed Self-organized Aggregation

Spreading and aggregation can be used together to produce distributed self-organized aggregation, i.e., computing in a distributed way the value of some properties of the system (i.e., the average temperature measured in the museum's sensor network) and have the results of such computation available at each and every node of the system, as from [Nath et al. 2004]. This mechanism is also at the basis of leader election algorithms via which to realize forms of distributed consensus, distributed task allocation and behavior differentiation [Bicocchi et al. 2012].

To ground the discussion, we illustrate how to trigger distributed self-organized aggregation of sensorial data. An employee in the museum wants to see in a display the maximum temperature sensed in the museum. In this example the `Aggregator` LSA is spread by a node to compute, in a distributed way, the maximum value of the "temp" property of LSAs available in the network (in this example, measured by a pool of sensors) and to show such a value on the display. For example the following LSA: `(property = temp; aggregation-op = max; diffusion-op = general; hop = 10)` would perform a distributed aggregation like the one depicted in Fig. 5.

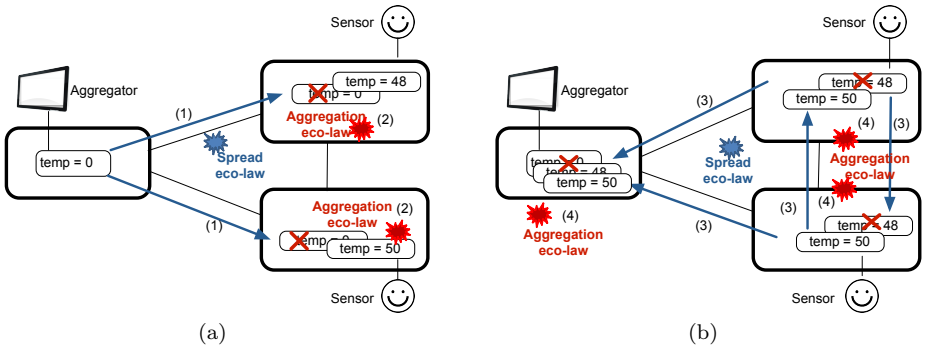


Fig. 5. Distributed aggregation. The `Aggregator` agent's LSA is spread over the network (a.1), to be aggregated with other LSAs (in this example from the `TemperatureSensor` agents) over the "temp" property (a.2). The result of aggregations is spread to other nodes (b.3) and then aggregated again (b.4) providing the `Aggregator` agent with the maximum "temp" value in the network. The final result can be presented on a suitable monitor display.

In particular, the assertion *aggregation_op = max* from one side aggregates LSAs around the property type "temp", from another side allows multiple copies of the same LSA to be aggregated and routed back to the source. Also in this case, all the complexity is moved at the middleware level via the eco-laws.

The same kind of approach can be used to realize gossip-based aggregation mechanisms [Bicocchi et al. 2012]: instead of propagating an LSA to all the neighbors, the LSA is sent only to a random subset of them. Probabilistically, this results in the same globally coherent behavior, but with less messages being exchanged.

In general, this kind of LSA aggregation supports situation-awareness in the system. In fact, it allows different information elements (e.g., sparse sensor readings) to be combined together into an higher-level representation of the current state in the environment (e.g., summary statistics on the sensed values).

5.5 Self-Organizing Spatial Coordination

Several different classes of self-organized spatial and motion coordination schemes, self-assembly, and distributed navigation can be expressed in terms of fields and pheromones.

In these approaches a number of field and pheromone data structures are laid out over the pervasive network. Agents coordinate their activities and movements on the basis of the local shape of such structures.

For example, field data structures can be used to coordinate the motion of people in the museum. Agents running on users' smart phones could give them directions by following the gradient of the fields in the environment. Considering the example in Figure 4, the agent could guide the user to the room by letting him/her follow the field of the room LSA (i.e., iteratively directing the user to the node associated with the *previous* variable). The strength of this approach resides in the fact that motion guidance is based on strictly local information (the local shape of the field) and is adaptive to changing conditions (e.g., a corridor made unaccessible) as the field would reshape accordingly.

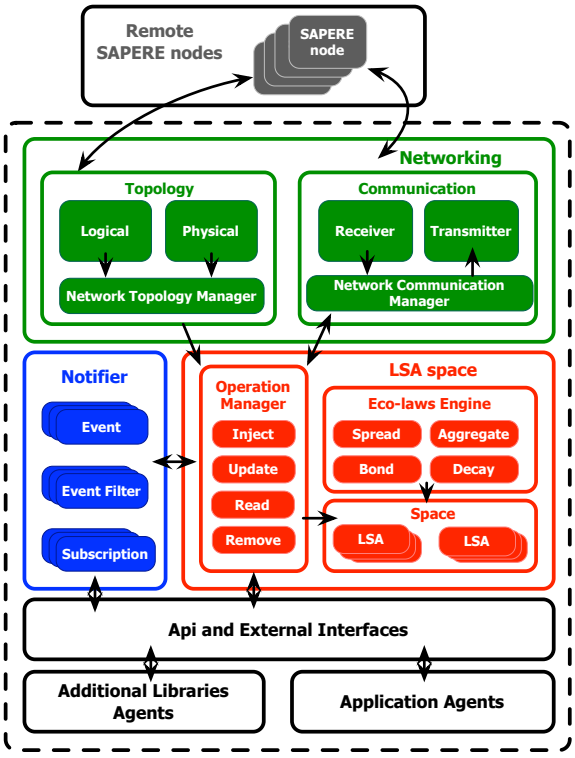


Fig. 6. The SAPERE Middleware Architecture

A similar approach can be used to allow different entities to leave pheromone trails in the museum to be followed later on by users or robots [Mamei and Zambonelli 2007]. A final relevant example consists of the spatial computing approach [Beal et al. 2012]. Also in this context, agent activities are driven by fields spread in an ad-hoc network of nodes. For example, a node can inject a field representing a query. Replying nodes can send a message (LSA) to the requestor by routing it following the gradient of the query field (this kind of gradient routing is often referred to as *chemotaxis*). Extending this principle, even more complex patterns and behaviors can be flexibly realized.

6. THE MIDDLEWARE IMPLEMENTATION

From an implementation-oriented viewpoint, the SAPERE middleware reifies LSAs in the form of tuples, to be dynamically stored and updated in a system of spatially-situated tuple spaces spread over the network devices.

On each node on which the SAPERE middleware is instantiated, it consists of three man components (see Figure 6):

—the *LSA space*, where local LSAs are stored and manipulated by the eco-laws;

—the *Notifier*, that manages events happening to LSAs and notifies the applications;

—the *Networking*, that builds and maintains the network topology, and rules the LSAs exchange with other nodes.

Possibly, additional libraries of agents could be deployed to enrich services offered by the middleware. In addition the SAPERE external interfaces, to ease the application programming, offer several programming agent templates facilitating the whole LSA life-cycle management. For instance the `SapereAgent` used in the code examples features seamless invocation of the middleware API.

6.1 The LSA space

The LSA space is realized as a lightweight tuple space that stores local LSAs and executes eco-laws over them. The core component is the *Space* that stores LSAs and provides access to them. The Operation Manager and the Eco-laws engine get mutual access to the Space to submit operations and execute eco-laws respectively over the LSAs collection.

The *Operation Manager* is used to submit operations to the Space, they are the operations that realize the basic SAPERE API (`injectLSA`, `updateLSA`, `onBond`, `onBondUpdate`). During the the operation execution the Operation Manager, for each submitted operation, interacts with the Notifier, adding or removing subscriptions to events and/or triggering new ones. Operations, in particular, are managed in two different ways: operations coming from external API are queued and passed to the Space one by one, operations coming from the Eco-laws Engine, instead, are executed as they arrive.

The *Eco-laws Engine* gets periodically activated to execute eco-laws on the collection of locally stored LSAs. To keep the bonds between LSAs consistent, a routine removing no longer valid bonds is run every time an update, or a remove operation is executed on the Space.

Therefore, at each wake-up the engine will process:

—non diffusive eco-laws in the following order: decay, aggregate, bond. That is, eco-laws that can possibly remove LSAs (e.g., the decay eco-law can remove an LSA that is decayed) are prioritized and managed in a finite state fashion, proceeding iteratively until applicable to a steady point in which no more aggregations are possible;

—the spread eco-law, executed once for each LSA requiring diffusion.

6.2 The Notifier

The Notifier component takes care of managing events happening to LSAs (e.g., bonds established, bonds removal, etc.) and notifying the Agents in charge of their management. This includes taking care of:

—*events to be notified*, for instance an `Event` related to the happening of a bond will be forwarded to one of the following methods: `onBond` to represent the happening of a new bond, `onBondUpdate` to represent the happening of some update in the content of an already bound LSA, `onUnbond` to represent the happening of the removal of a bond;

—*subscriptions*, to record the interest of a subscriber to a particular event. Subscriptions are managed automatically by the middleware and are not demanded

to the application programmer, in particular they are managed both by the OperationManager as operations are forwarded to the Space, and by the Agent that manages the LSAs for the `onBondUpdate`;

—*filters* responsible for discarding events not of interest for a subscriber. When an event occurs, the Notifier component determines which subscriber is interested in this event, applying the filter provided by the subscriber.

Events are fired by the Space object when operations on LSAs are performed. For each fired event, the Notifier checks if there are Subscriptions for that Event class and invokes the filters to detect the specific subscriber that shall be notified.

6.3 The Networking

The Networking module manages interactions with other SAPERE nodes. This implies two separate networking tasks:

—communicating with other nodes to exchange (spread) LSAs. To this end the *Network Communication Manager* provides two modules called respectively "Receiver" and "Transmitter" enabling communication between SAPERE nodes. Their actual implementation is based over standard Java Tcp/Ip sockets (however we have also tested over Bluetooth PAN/BNEP) and the Network Communication Manager might be easily extended to support other communication protocols;

—building and maintaining a topology of the network. In SAPERE, each node in the environment is made aware of one hop neighbors, and can communicate only with direct neighbors to exchange LSAs. This is realized by the local *Network Topology Manager*, which can easily support custom overlay networks by managing the resulting topology according to a specified policy. In particular, we have tested topologies based on both spatial proximity (defining an overlay network for nodes that are in a connectivity range as supported by the Bluetooth technology available on common smartphones) and on logical proximity (defining a overlay network that is based on the distance between entities linked to an external social network, e.g. Facebook). We have also considered mixed approaches as the result of crossing together the previous ones to identify as neighbors only those nodes that are in both social and physical proximity, as better described in [Zambonelli et al. 2011]..

The process of managing the topology of the network is distributed on each node, where the local Network Topology Manager, depending on the node configuration, instantiates a number of processes to build the network topology based on the chosen spatial model.

6.4 Launching SAPERE Applications

The SAPERE middleware has been developed in Java language, sources and binaries are available at *www.sapere-project.eu/download*. While on traditional personal computers SAPERE comes bundled over a Java Jar file to be linked to application-specific agent class, an optimized Apk bundle has been developed to be used on Android mobile devices.

Fig. 7. Screenshots from the Android SAPERE App. (left) The launch icon to be pushed on the Android springboard to start SAPERE and have that device to become a node of the ecosystem. (center) The user is presented a menu of the locally available SAPERE services; launching one of them brings to life the associated agents and their LSAs. (right) the GUI of a simple test application for inspecting the LSA space.

In general terms, launching the SAPERE App implies starting a local instance of the SAPERE middleware (which, depending on the configured networking strategy, will start looking for other SAPERE nodes to connect to, or will accept new connections) and presenting the users with a menu of the locally available SAPERE services. To add new services on a node, the developer has to produce the corresponding agent classes and link them to the SAPERE jar/bundle.

Figure 7 depicts and explains three exemplary menus from the Android version.

7. EXPERIMENTAL EVALUATION

The effectiveness of the SAPERE model and middleware has been evaluated along two main aspects: on the one hand, we performed a *software engineering* evaluation trying to assess whether SAPERE supports and facilitates the development of self-adaptive pervasive systems. On the other hand, we performed a *performance* evaluation trying to assess whether the SAPERE middleware can support a number of pervasive application with limited overhead.

7.1 Software Engineering Evaluation

To understand the effectiveness of the SAPERE model in developing pervasive applications, we tried to model some exemplary use-case applications both in SAPERE and in a FIPA-compliant agent-based middleware (MalacaTiny-Sol [Ayala et al. 2012]) in order to assess strengths and weaknesses of our system. In this section we report a specific case of a general analysis we described in [Ayala et al. 2013]. In particular, we model an emergency exit application in the smart museum scenario. In this application users are guided across the museum toward the closest/less crowded exit. It is assumed that a number of agents is running in the museum:

there are tourist agents associated to each visitor, and security agents associated to monitoring emergency and crowding conditions within the museum.

—*Design with SAPERE.* This modeling approach takes advantage of the uncoupled interactions among the agents. When an emergency is detected, security agents associated to exits propagate a LSA spreading across the building, and creating a field (or pheromone trail) leading to the exit. The shape of such a field can be influenced by the crowd distribution in the museum (computed via the aggregation eco-law). Tourist agents follow the resulting field downhill to reach the closest exit.

—*Design with FIPA Agents.* This modeling approach takes advantage of multicast communication among the agents. When an emergency is detected, the security agent notifies all the tourist agents about the situation. Tourist agents, provided with intelligent capabilities - and possibly requesting to other agents about the crowd distribution in the museum, plan the optimal route to the closest exit. An important strength of this system is that all the above behaviors can be flexibly encapsulated (e.g., via aspects or rule-based engines) to realize the agent code.

From a general perspective, this exemplary application highlights some key architectural features of SAPERE and FIPA agents. The differences of the two approaches for the development of pervasive application come not only from their schemas of interaction (LSA-spaces in SAPERE vs. message-passing in FIPA), but also from their mechanisms to support separation of concerns (i.e. eco-laws in SAPERE vs. agent functional behaviors in FIPA) and how they adapt the agent paradigm to pervasive computing (e.g. uncoupled interactions in SAPERE vs. multicast communication in FIPA).

More in detail, we highlight the following advantages of the two approaches.

—*SAPERE Advantages.* SAPERE agents have a good capacity for separation of concerns, components' decoupling and cohesion, and robustness. In the SAPERE LSA-based approaches - like in tuple spaces' - the service provision and consumption is more efficient than in FIPA ones because a direct interaction between agents is unnecessary. The negotiation is done in the LSA spaces via a pattern matching process that avoids message exchange. The distributed nature of SAPERE applications results in systems that adapt easily to changes in the physical space where the application is distributed. Finally, SAPERE spaces offers a more robust infrastructure thanks to its multiple SAPERE nodes. This is not a common feature in tuple based approaches but it is one of the strongest points of SAPERE.

—*FIPA Advantages.* In FIPA, the design of the agents is more scalable (in terms of complexity increase due to additional features) and can ensure the privacy of users more easily. In fact, FIPA-based approaches focus on the programming of agents' interval behavior. Accordingly they allow the set of services offered by the agents to be modified more easily. Some FIPA systems can do this at runtime because agents' behaviors can be encapsulated in aspects or sets of rules. Additionally, because the computing is encapsulated inside agents the risk of lateral effect when the system is extended is lower than in tuple based approaches and also in

SAPERE. Finally, in FIPA-based approaches it is somewhat easier to ensure the privacy of users, because most of the computation is performed locally.

Overall, the results from this analysis show that SAPERE effectively supports the development of adaptive pervasive applications with regard to interaction and distributed coordination, especially in the case of spatially-related tasks in which fields and distributed aggregation are most useful. From this perspective, we think that SAPERE fulfills its goal of supporting and facilitating adaptive pervasive systems.

Viceversa other platforms - like the considered FIPA-based system - better support agents' "internal" coding, agents planning and intelligent behaviors.

In general the two approaches could be combined with benefits to both. The functional behaviors of FIPA agents would simplify the deployment of complex agents in SAPERE. The main benefits for SAPERE would be to enhance the internal modularization of agents deployed in SAPERE nodes, to promote reuse and ease the adaptation of agents even at runtime.

7.2 Performance Evaluation

From the performances viewpoint, we have evaluated three main aspects: *(i)* local resource utilization *(ii)* time performances involved in running the eco-laws locally to a node; *(iii)* distributed overhead in supporting LSA spreading and aggregation across a network of SAPERE nodes.

From our investigations, the key parameter impacting on the performances of the SAPERE middleware is the number of LSAs populating the nodes. Indeed, the higher the number of LSAs stored in one node, the heavier can be the matching process involved in the LSA space engine to trigger eco-laws. Given the inherent local nature of LSA spaces, and the fact that the number of LSAs on one node expresses the number of local devices and service components on it – and to those eventually diffused from neighbor nodes – we assume that such a number will never grow excessively and set 2000 LSAs per node as an upper bound for our tests.

Experiments have been performed both on personal computers (Apple MacbookPro 2011 – Intel 2Ghz I7 processor – 8GB of RAM) and on mobile devices (Samsung Gt-P1000 running Android 2.3.6).

Local Resources Utilization. Table II illustrates local resources utilization of memory consumption and CPU usage at an increasing number of LSAs populating a single node, both on a personal computer and on an Android device.

With regard to the memory footprint, our measure highlights that:

—the memory occupancy of the bare SAPERE middleware is about 4KB;

—the memory occupancy of each LSA we developed for our experiments is on average 0.5 KB. This amount accounts for the LSA object instance itself, the LSA's payload (that is application specific, in these experiments contains a temperature measure) and the `SapereAgent` instance managing the LSA's life cycle.

Our measures suggest that for reasonable numbers of LSAs per node (a mobile unit will unlikely host more than a dozen of local services and sensors) the SAPERE middleware is lightweight enough to be hosted on even small devices.

The memory footprint size has been estimated using the SizeOf library (*sizeof.*

Table II. Local resources utilization on a node against the number of locally stored LSAs

LSAs	M. Footprint	PC CPU %	Mobile CPU %
0	4 KB	1%	24%
100	55 KB	4%	30%
200	111 KB	5%	60%
500	249 KB	7%	75%
1000	488 KB	9%	90%
2000	1002 KB	18%	100%

sourceforge.net). The CPU usage has been measured with Oracle JConsole (included in the Oracle JDK) and Google DDMS (included in the Google Android JDK) on personal computers and Android mobile devices, respectively.

Time Performances. We analyze the time performances associated to the execution of the SAPERE eco-laws. For each of the experiments we executed 1000 runs on a personal computer and averaged the results. Similar trends, although with a reduced number of runs, have been obtained also on Android devices.

Bond Eco-Law. We measured the time required to realize a bond between 2 LSAs. We run the experiments by pre-injecting a number of LSAs and then by injecting an LSA with the "?" formal value: Figure 8(a) reports the time occurring between the injection of the last LSA and the notification of the bond event. For a number of pre-injected LSAs lower than 2000, the binding time is below 10ms. The time required grows linearly with the number of the involved LSAs since the current LSA engine is not provided of a mechanism for LSAs' indexing and quick verification of pattern-matching. However, the performances appear perfectly acceptable with respect to its reference context, in that completely compatible with average human reaction time and also with the frequencies of contextual changes.

Aggregation Eco-Law. We measured the time required to aggregate a subset of the LSAs populating the space. Results of this experiment (aggregating the 25%, the 50% and the 75% of the LSAs in a node) are in Figure 8(b): the time required by the aggregation is not affected by the percentage of the LSAs to be aggregated but by the whole number of LSAs populating the space. Again, the lack of an indexing mechanism for LSAs makes the execution of the aggregation eco-law to require an exhaustive search over the whole LSAs space. Yet, the overall time is still in line with the expectation of a large class of interactive application-level user services.

Spread Eco-Law. We measured the round-trip time required to spread an LSA from a source node to a destination node and to spread it back to the source, using Wi-Fi connections as a career. The key parameters of this experiment are: (1) the distance in hop-terms between the source and the destination, (2) the number of LSAs populating each node. Results in Figure 8(c) show that the time linearly increases with the hop distance, as the LSA needs to move across multiple nodes.

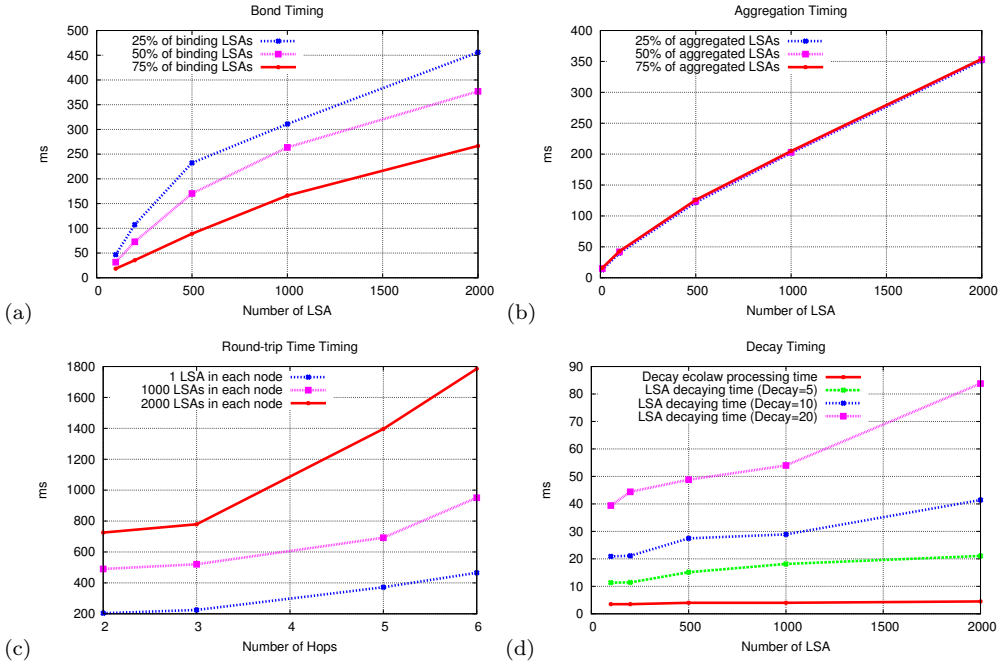


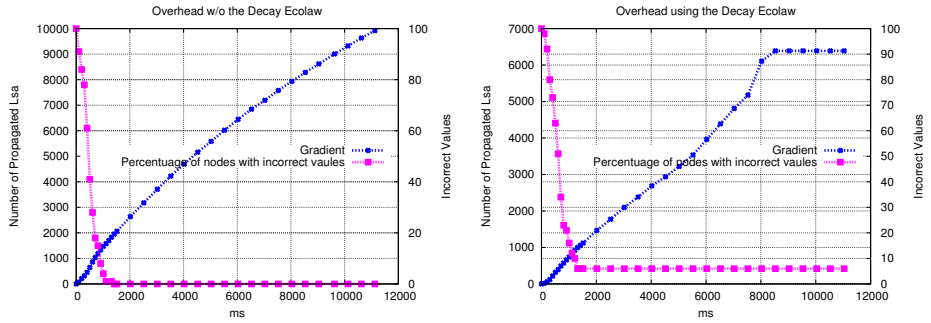
Fig. 8. (a) Time for binding two LSAs. (b) Aggregation time with a growing number of LSAs populating the space.(c) Round-trip time to spread an LSA to a remote node and to get a reply back. (d) Time to decay a given LSA on the basis of the number of LSAs populating the space.

It also linearly increases with the number of LSAs because of the exhaustive search applied to the LSAs population to detect those to be propagated. Although a bit higher, and highly influenced by WiFi performances, also in this case the time required appears acceptable from the application viewpoint.

Decay Eco-Law. In this experiment we measure the time required to remove an expired LSA. For a varying number of LSA, we set one of them to be decayed after 5, 10 or 20 middleware cycles. Results of this experiment are in Figure 8(d) and are in line with the other results involving an exhaustive search over the LSAs.

Distributed Overhead Evaluation. We measured the number of operations required to propagate information as a field across SAPERE nodes. That involves spreading LSAs from node to node and aggregating those multiple copies getting the same node from different sources. Such a process requires time to converge and repeated aggregations. In addition, to ensure the coherency of the field structure despite network dynamism, the spread has to be periodically repeated.

Figure 9(a) shows the number of operations required to spread a field in a network (a regular lattice with connectivity grade 4) of 100 SAPERE nodes tracing the number of LSAs exchanged. The field quickly stabilizes, while the constant increment of operations involved, even after convergence, is due to the field periodic refresh. Figure 9(b) shows the effect of introducing a decaying factor in the field: as soon as the decaying procedure completes, the field is removed from the



(a) (b)

Fig. 9. (a) Number of LSA exchanges required to propagate and maintain the field distributed shape. (b) Number of LSA exchanges required once a decaying factor is introduced.

network and the number of operations drops to zero (thus allowing multiple fields to be spread in the network without affecting performances in the long run).

Summarizing, the reported experimental results show that our current SAPERE implementation – despite being highly un-optimized – is already compatible with a large class of pervasive computing applications. In particular:

—The memory footprint and CPU usage have a limited impact enabling the effective deployment of SAPERE over mobile devices;

—The time for triggering and executing eco-laws allows for application-level reaction times compatible with interactive pervasive applications;

—The time for spreading LSAs across a network is acceptable and compatible with application-level expectations, and its overhead on the network is limited.

8. RELATED WORK

In the past few years, several proposals have tried to identify novel models and mechanisms to support the design and development of pervasive service systems.

Many approaches, starting from service-oriented architectures [Huhns and Singh 2005], in the attempt to overcome the static and context-unaware mechanisms of service discovery and composition, propose innovative mechanisms [Kalasapur et al. 2007; Bronsted et al. 2010] and associated middleware infrastructures [Raychoudhury et al. 2013]. These include novel approaches to context-aware discovery and novel means to handle the dynamic arrival and dismissing of service components [Bronsted et al. 2010] – there possibly included service re-location [Riva et al. 2007] – and novel mechanisms to evaluate at run-time the most suitable service composition patterns depending on current context and availability of components [Kalasapur et al. 2007]. SAPERE, with the single mechanism of bonding, supports both dynamic context-aware service discovery and composition. Also, SAPERE can support self-organizing patterns of distributed service compositions and orchestration, and thus achieve high degree of adaptivity with little programming efforts.

Different threads of research explore solutions based on coordination modes promoting weaker degree of coupling between service components than service-oriented

architectures, e.g., event-based coordination models and tuple-based ones [Eugster et al. 2003]. In particular, tuple-based coordination models, by expressively enabling both flexible communication and synchronization of activities, have been extensively exploited as a tool to coordinate service activities in mobile environments. Proposals typically rely on networked tuple spaces spread across pervasive environments and mobile devices [Bellavista et al. 2012; Murphy et al. 2006; De Nicola et al. 2014], possibly integrating mechanisms to dynamically re-configure the standard pattern-matching mechanisms of tuple spaces [Omicini and Zambonelli 1999]. SAPERE has been definitely inspired by tuple-space coordination models, but has radically re-defined it with its concepts of LSAs and eco-laws.

TOTAM [Harnie et al. 2014] is a tuple-based infrastructure that addresses scenarios of pervasive computing very similar to those of SAPERE. In addition, again similarly to SAPERE, it proposes exploiting sort of enhanced tuple spaces associated to pervasive devices to support localized spatial interactions in urban scenarios, as well as information diffusion and propagation. However, unlike SAPERE, TOTAM does not deal with the issue of supporting adaptive and self-organizing coordination patterns, leaving up to application agents the duty of organizing their own coordination schemes.

The concept of LSA of SAPERE shares some key characteristics and goals with the concept of dynamic tuples proposed in [Stovall and Julien 2008], i.e., the idea that dynamically changing fields in tuples can support adaptive context-aware discovery of services. However, the LSA concept of SAPERE is embedded in a fully-fledged framework where this concept finds practical and complete realization.

The TOTA system [Mamei and Zambonelli 2009], previously developed within our research group, shares the idea of SAPERE of enabling the flexible programming of self-organizing distributed coordination schemes. However, TOTA was capable of supporting only self-organization mechanisms based on distributed computational fields. SAPERE has a more general nature, being capable of supporting fields but also more general mechanisms and schemes of self-organization, such as pheromones, gossip schemes, distributed aggregation. Similar considerations apply to other spatial computing models based on computational fields, like Proto [Beal et al. 2012].

The concept of augmented ecologies [Tisato et al. 2012] shares with SAPERE both the ecosystem inspiration and the idea of mapping components of a pervasive environment into a virtual ecosystem of organisms interacting in a spatial and context-dependent way. However, the augmented ecologies proposal does not aim at supporting some specific adaptive coordination model, leaving up to application components the duty of ruling their own coordination activities.

Chemical bonds have been proposed as a middleware mechanism to promote spontaneous service and workflow in open environments [Banâtre and Priol 2009; Fernandez et al. 2014]. Unlike SAPERE, though, such proposals do not integrate the spatial mechanism required for decentralized pervasive environments. In a recent work [Viroli et al. 2011], we have proposed a chemically-inspired coordination model that, although not being coupled with a specific eco-laws model or implemented middleware, can be considered our first attempt towards nature-inspired pervasive service coordination.

9. CONCLUSIONS AND FUTURE WORK

SAPERE proposes a radically new approach to engineer pervasive computing services that suits the emerging characteristics of pervasive computing environments. In particular:

—Its nature-inspired coordination model based supports spontaneous, context-aware, and adaptive interactions among situated pervasive service components;

—A variety of adaptive self-organizing patterns can be enforced in SAPERE, to realize several effective schemes for the provisioning of distributed pervasive services;

—The SAPERE middleware effectively supports the SAPERE model and a variety of networking schemes with acceptable performances.

Currently, we are working to improve some implementation aspects of the SAPERE middleware, in particular to optimize LSAs storing and access, and to extend its support for semantic data representation [Stevenson et al. 2012]. As a plan for future work, we intend to experience the SAPERE approach with a number of innovative services in the area of urban computing [Harnie et al. 2014; Zambonelli 2012] and smart mobility services [Riener and Ferscha 2013].

Acknowledgment

Work supported by the SAPERE (Self- Aware Pervasive Service Ecosystems) project (EU FP7-FET, Contract No. 256873).

REFERENCES

Alt, F., Schmidt, A., and Mü andller, J. 2012. Advertising on public display networks. *Computer 45,* 5, 50 –56.

Ayala, I., Amor, M., and Fuentes, L. 2012. Self-configuring agents for ambient assisted living applications. *Personal and Ubiquitous Computing*, 1–11.

Ayala, I., Amor, M., Fuentes, L., Mamei, M., and Zambonelli, F. 2013. Developing pervasive agent-based applications: A comparison of two coordination approaches. In *International Workshop on Agent-Oriented Software Engineering, LNCS 7852*.

Babaoglu, O., Canright, G., Deutsch, A., Caro, G. A. D., Ducatelle, F., Gambardella, L. M., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., and Urnes, T. 2006. Design patterns from biology for distributed computing. *ACM Transactions on Autonomous and Adaptive Systems 1,* 1, 26–66.

Banâtre, J.-P. and Priol, T. 2009. Chemical programming of future service-oriented architectures. *Journal of Software 4,* 7, 738–746.

Beal, J., Dulman, S., Usbeck, K., Viroli, M., and Correll, N. 2012. Organizing the aggregate: Languages for spatial computing. In *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*.

Bellavista, P., Corradi, A., Fanelli, M., and Foschini, L. 2012. A survey of context data distribution for mobile ubiquitous systems. *ACM Computing Surveys 4,* 44.

Bicocchi, N., Mamei, M., and Zambonelli, F. 2012. Self-organizing virtual macro sensors. *ACM Transaction on Autonomous Adaptive Systems 7,* 1.

Brazier, F. M., Kephart, J. O., Parunak, H. V. D., and Huhns, M. N. 2009. Agents and service-oriented computing for autonomic computing: A research agenda. *IEEE Internet Computing 13,* 3, 82–87.

Bronsted, J., Hansen, K., and Ingstrup, M. 2010. Service composition issues in pervasive computing. *IEEE Pervasive Computing 9,* 1, 62–70.

Campbell, A. T., Eisenman, S. B., Lane, N. D., Miluzzo, E., Peterson, R. A., Lu, H., Zheng, X., Musolesi, M., Fodor, K., and Ahn, G.-S. 2008. The rise of people-centric sensing. *IEEE Internet Computing 12,* 4.

Cheng, B. et al. 2009. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, B. Cheng, R. d. Lemos, P. Inverardi, and J. Magee, Eds. Lecture Notes in Computer Science, vol. 5525. Springer, 1–26.

de Lemos, R. et al. 2013. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems*. Lecture Notes in Computer Science, vol. 7475. Springer, 1–32.

De Nicola, R., Pugliese, R., and Tiezzi, F. 2014. Programming autonomic service ensembles: the scel language. *ACM Transactions on Autonomous and Adaptive Systems 9,* 2.

Elhart, I., Langheinrich, M., Davies, N., and José, R. 2013. Key challenges in application and content scheduling for open pervasive display networks. In *IEEE International Conference on Pervasive Computing and Communications Workshops, San Diego, CA*. 393–396.

Eugster, P., Felber, P., Guerraoui, R., and Kermarrec, A. 2003. The many faces of publish/subscribe. *ACM Computing Surveys 35,* 2, 114 – 131.

Fernandez, H., Tedeschi, C., and Priol, T. 2014. Rule-driven service coordination middleware for scientific applications. *Future Generation Comping Systems 35*, 1–13.

Gelernter, D. 1985. Generative communication in linda. *ACM Transactions on Programming Languages and Systems 7,* 1 (Jan.), 80–112.

Greenberg, S., Marquardt, N., Ballendat, T., Diaz-Marino, R., and Wang, M. 2011. Proxemic interactions: the new ubicomp? *ACM Interactions 18,* 1.

Gu, Y., Lo, A., and Niemegeers, I. 2009. A survey of indoor positioning systems for wireless personal networks. *IEEE Communications Surveys and Tutorials 11,* 1, 13–32.

Harnie, D., D'Hondt, T., Boix, E. G., and De Meuter, W. 2014. Programming urban-area applications for mobility services. *ACM Transactions on Autonomous and Adaptive Systems 9,* 2.

Holldobler, B. and Wilson, O. 2009. *The Superorganism: the Beauty, Elegance, and Strangeness of Insect Societies.* W. W. Norton and C., New York, NY.

Huhns, M. N. and Singh, M. P. 2005. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing 9,* 1, 75–81.

Jelasity, M., Montresor, A., and Babaoglu, Ö. 2005. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems 23,* 3, 219–252.

Kalasapur, S., Kumar, M., and Shirazi, B. A. 2007. Dynamic service composition in pervasive computing. *IEEE Transactions on Parallel and Distributed Systems 18,* 7 (July), 907–918.

Kephart, J. O. and Chess, D. M. 2003. The vision of autonomic computing. *IEEE Computer 36,* 1, 41–50.

Lukowicz, P., Pentland, S., and Ferscha, A. 2012. From context awareness to socially aware computing. *IEEE Pervasive Computing 11,* 1, 32 –41.

Mamei, M. and Zambonelli, F. 2007. Pervasive pheromone-based interaction with rfid tags. *ACM Transactions on Autonomous and Adaptive Systems 2,* 2, 1–28.

Mamei, M. and Zambonelli, F. 2009. Programming pervasive and mobile computing applications: the tota approach. *ACM Transactions on Software Engineering and Methodology 18,* 4.

Murphy, A. L., Picco, G. P., and Roman, G.-C. 2006. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology 15*, 279–328.

Nath, S., Gibbons, P. B., Seshan, S., and Anderson, Z. R. 2004. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 250–262.

Omicini, A. 2012. Nature-inspired coordination for complex distributed systems. In *Intelligent Distributed Computing VI*. Studies in Computational Intelligence, vol. 446. Springer, 1–6.

Omicini, A. and Zambonelli, F. 1999. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems 2,* 3, 251–269.

Parunak, V. 1997. Go to the ant: Engineering principles from natural multi-agent systems. *Annals of Operations Research 75*, 69 – 101.

Pejovic, V. and Musolesi, M. 2013. Anticipatory Mobile Computing: A Survey of the State of the Art and Research Challenges.

Raychoudhury, V., Cao, J., Kumar, M., and Zhang, D. 2013. Middleware for pervasive computing: A survey. *Pervasive and Mobile Computing 9,* 2, 177 – 200.

Read, M., Andrews, P. S., and Timmis, J. 2012. An introduction to artificial immune systems. In *Handbook of Natural Computing*. Springer, 1575–1597.

Riener, A. and Ferscha, A. 2013. Enhancing future mass ict with social capabilities. In *Co-evolution of intelligent socio-technical systems*. Understanding Complex Systems. Springer Berlin Heidelberg, 141–184.

Riva, O., Nadeem, T., Borcea, C., and Iftode, L. 2007. Context-aware migratory services in ad hoc networks. *IEEE Transactions on Mobile Computing 6*, 1313–1328.

Roggen, D., Tröster, G., Lukowicz, P., Ferscha, A., del R. Millán, J., and Chavarriaga, R. 2013. Opportunistic human activity and context recognition. *IEEE Computer 46,* 2, 36–45.

Schuster, D., Rosi, A., Mamei, M., Springer, T., Endler, M., and Zambonelli, F. 2013. Pervasive social context: Taxonomy and survey. *ACM Transactions on Intelligent Systems and Technology 4,* 3.

Stevenson, G., Viroli, M., Ye, J., Montagna, S., and Dobson, S. 2012. Self-organising semantic resource discovery for pervasive systems. In *1st International Workshop on Adaptive Service Ecosystems: Natural and Socially Inspired Solutions*. Lyon, France, 47–52.

Stovall, D. and Julien, C. 2008. Rapid prototyping of routing protocols with evolving tuples. In *Distributed Applications and Interoperable Systems, 8th IFIP WG 6.1 International Conference, DAIS 2008, Oslo, Norway, June 4-6, 2008. Proceedings*. Lecture Notes in Computer Science, vol. 5053. 296–301.

Tisato, F., Simone, C., Bernini, D., Locatelli, M. P., and Micucci, D. 2012. Grounding ecologies on multiple spaces. *Pervasive and Mobile Computing 8,* 4 (Aug.), 575–596.

Viroli, M., Casadei, M., Montagna, S., and Zambonelli, F. 2011. Spatial coordination of pervasive services through chemical-inspired tuple spaces. *ACM Transactions on Autonomous and Adaptive Systems*.

Vromant, P., Weyns, D., Malek, S., and Andersson, J. 2011. On interacting control loops in self-adaptive systems. In *2011 ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems, Honolulu , HI*. ACM, 202–207.

Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., and Göschka, K. 2012. On patterns for decentralized control in self-adaptive systems. *Software Engineering for Self-Adaptive Systems II*, 76–107.

Ye, J., Dobson, S., and McKeever, S. 2012. Situation identification techniques in pervasive computing: A review. *Pervasive and Mobile Computing 8,* 1, 36–66.

Zambonelli, F. 2012. Toward sociotechnical urban superorganisms. *IEEE Computer 47,* 8.

Zambonelli, F., Castelli, G., Mamei, M., and Rosi, A. 2011. Integrating pervasive middleware with social networks in sapere. In *Mobile and Wireless Networking (iCOST), 2011 International Conference on Selected Topics in*. 145 –150.

Zambonelli, F. and Viroli, M. 2011. A survey on nature-inspired metaphors for pervasive service ecosystems. *Journal of Pervasive Computing and Communications 7*, 186–204.