

Programming Self-organizing Pervasive Applications with SAPERE

Franco Zambonelli, Gabriella Castelli, Marco Mamei, and Alberto Rosi

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”. Finally, we show how in SAPERE one can express general-purpose distributed self-organizing schemes.

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 the 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 [12]. Unfortunately, most of the solutions so far proposed are in terms of “add-ons” to be integrated in existing frameworks [1]. The result is often an increased complexity of current frameworks and the emergence of contrasting trade-off between different solutions.

In our opinion, there is need for tackling the problem at the foundation, conceiving a radically new way of modeling integrated pervasive services and their ex-

Dipartimento di Scienze e Metodi dell’Ingegneria,
University of Modena and Reggio Emilia
e-mail: name.surname@unimore.it

cution environments, such that the 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 (www.sapere-project.eu), which proposes a novel nature-inspired approach to support the design and development of adaptive and self-organizing systems of pervasive computing services.

In this context, the contribution of this paper is twofold: (i) We present the overall conceptual architecture of the SAPERE approach, and show how it has been realized in the SAPERE middleware; (ii) We detail the specific approach to distributed self-organizing coordination promoted by SAPERE and discuss how this supports the effective development and execution of self-organizing pervasive applications.

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 distributed computational *ecosystem*.

As 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 the 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 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 collect 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.

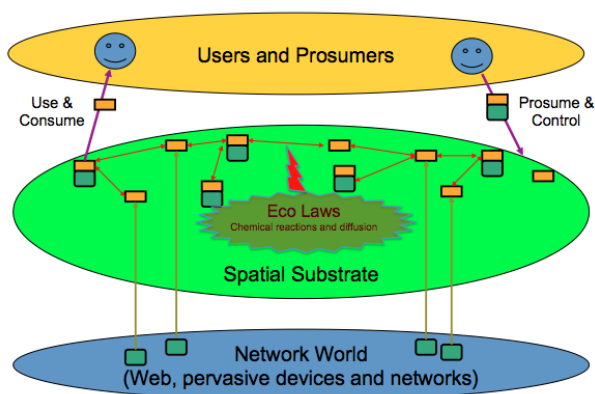


Fig. 1 The SAPERE Reference Architecture

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 the ambition of 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 [11] 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 provides the same set of functions.

Each SAPERE node hosts a local tuple space [2], that acts as a local repository of LSAs for local agents, and a local eco-laws engine. The LSA-space of each node is in 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. 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 applies to rule the 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 of an LSA to keep it “alive” – `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. E.g., 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 [12].

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 [9]. Here we emphasize that, by building over tuple-based models and extending upon them [2], 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 the 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 match with each other. The properties present in LSAa (e.g., accuracy) are not taken into account by the matching function because it considers only inclusive match.

4 The Eco-laws Set

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

4.1 Bonding

Bonding is the primary form of 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 holding.

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 parameterized 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 [3], 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.

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's 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 [7].

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 the vanishing of components 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 are the result of 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 by 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 the specific type of propagation.

Two different types of propagation are implemented in the SAPERE framework: (i) a direct propagation used to spread an LSA to a specified neighbor node, e.g., `LSAe:(...diffusion_op=direct; destination=node_x; ...)`; (ii) 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.

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 the necessary phases of discovery and invocation of traditional service systems. Spreading is necessary in that there must be a mean to diffuse information in a distributed environment to enable distributed interactions. Aggregation and decay are necessary to support 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”), there included 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 [1].

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, also known as *computational force fields*. As detailed in [5], many different classes of self-organized motion coordination schemes, self-assembly, and distributed navigation can be expressed in terms of gradients. For instance, Figures 2 shows how it is possible to define a “Guide” agent that builds, with its LSA, a distributed computational field and another agent “Search” that follows such field uphill.

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

```

Agent Guide {
  init() { injectLSA(name = guide, diffusion_op = general,
    hop = 1, aggregation_op = min, previous = local) } }

Agent Search {
  init() { injectLSA(name = guide, hop = *) }

  onBond(LSA b) { float d = computeDistanceFromHop(b.hop)
    print("guide distance = "+d)
    print("go toward "+b.previous) } }

```

Fig. 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.

```

Agent X {
  init() { injectLSA(aggregation_op = max, property = temp, diffusion = general,
    hop = 1, previous = local) } }

Sensor 1...N {
  init() { float t = sample()
    injectLSA(temp = t) }

  run() { while(true) { float t = sample()
    updateLSA(temp = t) }}}

```

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

and every node of the system, as from [7]. 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 how it is possible to aggregate temperature information from multiple distributed sensors.

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 [1]. 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.

6 Related Works and Conclusions

While exploiting a number of common features with situated pervasive approaches based on tuple spaces [6, 5, 4], SAPERE proposes a different concept and management of space. Indeed local spaces in SAPERE cannot merge – as the ones in *Lime* [6] do. Differently from TOTA [5] the behaviors are embedded in the eco-laws rather than in the tuples. Moreover components in a SAPERE environment do not need

to declare a personalized view of the context as in *Ego-spaces* [4], that is naturally provided by injecting their own LSAs.

Some approaches, such as *SwarmLinda* [10], exploit the properties of adaptive self-organizing natural systems to enforce adaptive behaviors transparently to applications. SAPERE has the more ambitious goal of promoting and easing the programming of self-organizing distributed behaviors.

Summarizing, the innovative nature-inspired approach of SAPERE is effective to easily enforce a variety of self-organizing schemes for pervasive computing services. As the activities within the SAPERE European Project will proceed, we will challenge the SAPERE findings and tools against innovative services in the area of crowd management, by exploiting an ecosystem of pervasive displays as a technical testbed [8].

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

References

1. Ozalp Babaoglu and al. Design patterns from biology for distributed computing. *ACM Trans. Auton. Adapt. Syst.*, 1(1):26–66, 2006.
2. David Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, January 1985.
3. Michael N. Huhns and Munindar P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.
4. Christine Julien and Gruia-Catalin Roman. Egospaces: Facilitating rapid development of context-aware mobile applications. *IEEE Trans. Software Eng.*, pages 281–298, 2006.
5. Marco Mamei and Franco Zambonelli. Programming pervasive and mobile computing applications: the tota approach. *ACM Trans. Software Engineering and Methodology*, 18(4), 2009.
6. Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Trans. Software Engineering and Methodology*, 15(3):279–328, 2006.
7. Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 250–262, Baltimore, MD, USA, 2004.
8. Andreas Sippl, Clemens Holzmann, Doris Zachhuber, and Alois Ferscha. Real-time gaze tracking for public displays. In *First International Joint Conference on Ambient Intelligence*, volume 6439 of *Lecture Notes in Computer Science*, pages 167–176. Springer, 2010.
9. Graeme Stevenson, Mirko Viroli, Juan Ye, Sara Montagna, and Simon Dobson. Self-organising semantic resource discovery for pervasive systems. In *1st International Workshop on Adaptive Service Ecosystems: Natural and Socially Inspired Solutions*, pages 47–52, Lyon, France, 2012.
10. Robert Tolksdorf and Ronaldo Menezes. Using swarm intelligence in linda systems. *Lecture Notes in Computer Science*, pages 519–519, 2004.
11. Franco Zambonelli, Gabriella Castelli, Marco Mamei, and Alberto Rosi. Integrating pervasive middleware with social networks in sapere. In *International Conference on Selected Topics in Mobile and Wireless Networking*, pages 145–150, Shanghai, PRC, 2011.
12. Franco Zambonelli and Mirko Viroli. A survey on nature-inspired metaphors for pervasive service ecosystems. *Journal of Pervasive Computing and Communications*, 7:186–204, 2011.