Injecting Self-Organisation into Pervasive Service Ecosystems

Sara Montagna, Mirko Viroli, Jose Luis Fernandez-Marquez, Giovanna Di Marzo Serugendo & Franco Zambonelli

Mobile Networks and Applications
The Journal of SPECIAL ISSUES on Mobility of Systems, Users, Data and Computing

ISSN 1383-469X

Mobile Netw Appl
DOI 10.1007/s11036-012-0411-1

Springer

Mobile Netw Appl
DOI 10.1007/s11036-012-0411-1

Injecting Self-Organisation into Pervasive Service Ecosystems

Sara Montagna · Mirko Viroli · Jose Luis Fernandez-Marquez · Giovanna Di Marzo Serugendo · Franco Zambonelli

Abstract Pervasive service ecosystems are a new paradigm for the design of context-aware systems featuring adaptivity and self-awareness. A theoretical and practical framework has been proposed for addressing these scenarios, taking primary inspirations from natural ecosystems and grounding upon two basic abstractions: "live semantic annotations" (LSAs), which are descriptions stored in infrastructure nodes and wrapping data, knowledge, and activities of humans, devices, and services; and "eco-laws", acting as system rules evolving the population of LSAs as if they were molecules subject to chemical-like reactions. In this paper, we aim at deepening how self-organisation can be injected in pervasive service ecosystems in terms of spatial structures and algorithms for supporting the design of context-aware applications. To this end, we start from an existing classification of self-organisation patterns, and systematically show how they can be supported in pervasive service ecosystems, and be composed to generate a self-organising emergent behaviour. A paradigmatic crowd steering case study is used to demonstrate the effectiveness of our approach.

Keywords Self-organising systems · Bio-inspired mechanisms · Pervasive computing · Context awareness

This work has been supported by the EU-FP7-FET Proactive project SAPERE Self-aware Pervasive Service Ecosystems, under contract no.256873.

S. Montagna (✉) · M. Viroli
Alma Mater Studiorum–Università di Bologna,
Via Venezia 52, 47521 Cesena, Italy
e-mail: sara.montagna@unibo.it

M. Viroli
e-mail: mirko.viroli@unibo.it

J. L. Fernandez-Marquez · G. Di Marzo Serugendo
University of Geneva, Battelle, Batiment A, Route de Drize 7, 1227 Carouge, Switzerland

J. L. Fernandez-Marquez
e-mail: joseluis.fernandez@unige.ch

G. Di Marzo Serugendo
e-mail: giovanna.dimarzo@unige.ch

F. Zambonelli
Università di Modena e Reggio Emilia,
Via G. Amendola 2, 42122 Reggio Emilia, Italy
e-mail: franco.zambonelli@unimore.it

1 Introduction

A *pervasive service ecosystem* is a computing system immersed in our everyday environment, made of actors and components of various kinds, which we refer to as *individuals*. They can be mobile users, their smartphones, software services, pervasive displays, sensors and devices spread across the environment, sources of knowledge, data and events. They all interoperate opportunistically to achieve individual goals, but are also globally guided and governed by some "laws" enacted by the infrastructure. The pervasive service ecosystem (pervasive ecosystem in short) paradigm is aimed at tackling context-awareness of components and processes, and handling adaptivity without human or supervised control. This scenario has notable relations with natural ecosystems, and, in particular, it shares with them the need of an intrinsic self-organisation by which the globally intended behaviour emerges out of the local interaction of individuals, driven by the local manipulations enacted by those

Published online: 16 September 2012

laws. Examples of application contexts include provisioning of visualisation services on adaptive displays, services for smart cities, intelligent traffic control, and augmented social reality [41].

To address the engineering of this kind of systems, which will soon emerge due to the increasingly widespread diffusion of pervasive computing technologies, we adopt a framework of self-organising coordination [35] tailored to spatially distributed, context-dependent, and open systems—as pervasive computing systems are. Drawing on existing works in the field of coordination models and languages [6, 33, 34, 36], middlewares for context-aware applications [5, 16, 24], and on models of concurrency and bio-inspired computing [9, 22, 25], we adopt a model based on the idea of continuously reflecting the presence and activities of components in the pervasive computing system through *Live Semantic Annotations* (LSAs), which are stored across computational devices, and altogether form a global network of annotations representing the virtual counterpart of the ecosystem. Ecosystem behaviour is regulated by a set of laws, called *eco-laws*, which act locally on each node and its neighbourhood, combining and manipulating annotations using a chemical style and semantic pattern matching [37, 39].

In this paper we study the problem of supporting self-organising mechanisms on top of this model. Self-organisation is known to be an effective source of mechanisms for supporting applications that require adaptivity and toleration of unpredictable changes, by ensuring that spatial and temporal patterns of behaviour emerge out of local interactions and without a central authority that imposes pre-defined plans. Such patterns are usually inspired by natural systems, and show appealing characteristics for pervasive scenarios, since they are, first of all, able to adapt to environmental changes and able to achieve complex behaviours using a limited set of basic rules [10].

To this end, we adopt the classification of self-organising patterns provided in [11], in which a set of basic functioning behaviours is discussed which can be used in isolation or composed together to form more complex patterns. In order to equip a pervasive ecosystem with self-organisation we here propose a set of eco-laws enacting fine-grained processes of LSA diffusion, interaction, composition and decay: they will be shown not only to support basic self-organising patterns, but also their seamless composition into more articulated ones.

The proposed approach is exemplified by a pervasive computing scenario of crowd steering, in which groups of people are guided (by signs appearing in public/private displays) towards locations based on their preference, along optimal paths and taking into account contextual information describing the presence of crowded areas which should be dynamically intercepted and circumvented.

The remainder of the paper is organised as follows: Section 2 introduces the pervasive service ecosystem framework (detailing the outlined version appeared in [37] and refining the eco-law language presented in [39]), Section 3 discusses the problem of injecting self-organisation in it, Section 4 presents the corresponding solution we propose, Section 5 presents the case study, Section 6 discusses related works, and finally Section 7 concludes providing final remarks.

2 Scenario, requirements, motivations and concepts

One notable application scenario of pervasive ecosystems is that of crowd steering, which we here introduce to describe the general requirements of situation-awareness and adaptivity that our approach faces. The idea is to guide people towards locations hosting events of interest in a complex and dynamic environment (using semantic matching with people's interests), avoiding obstacles that can dynamically appear and disappear, such as crowded rooms or corridors, and without any supervision—namely, in a self-organised way. In particular, we consider a museum with a set of rooms connected by corridors, whose floor is covered with a network of computational devices (sensor nodes). These devices exchange information with each other based on proximity, sense the presence of visitors, and hold information of various kinds (e.g., about exhibits currently active in the museum). Visitors exploring the museum are equipped with a smartphone that holds their preferences. By interaction with sensor nodes, a visitor can be guided towards rooms with a target matching his/her interest, thanks to signs dynamically appearing on the smartphone or on public displays.

2.1 Pervasive system requirements

Pervasive ecosystems deal with spatially-, temporally- and socially-situated activities of users, and should therefore be able to interact with the surrounding world and adapt their behaviour accordingly. In the museum infrastructure, local information from crowd sensors has to be exploited and propagated around to create a global awareness of crowd distribution over the exposition. *Situation-awareness* is hence a key requirement, and it is generally achieved by infrastructures reifying

data/knowledge/events in the precise point (or region) of space where they pertain, promoting interactions just based on proximity.

Another complementary requirement is *adaptivity*: pervasive ecosystems and their infrastructures should inherently exhibit properties of autonomous adaptation and management to survive contingencies without human intervention and/or global supervision. In the museum infrastructure, the direction towards the point of interest has to be defined according to the current state of the surrounding physical and social environment, e.g., circumventing dynamically forming crowded places. Namely, when new devices are deployed, new information is injected, or new people arrive, a spontaneous re-distribution and re-shaping of the overall system information should take place. For instance: the route towards an exhibition could be automatically computed by self-organisation so as to dynamically avoid overcrowded rooms or corridors, or alternative exhibitions may be selected if one has reached (or will shortly reach) its maximum capacity. Adaptivity is often achieved by designing coordination rules that by acting locally (namely, on a given network neighbourhood) make global properties emerge dynamically—following e.g. a natural inspiration as in [36].

Finally, to support *openness*, pervasive ecosystems should feature standard technologies for the description of services, and for the rules by which we manipulate such descriptions. This has the twofold goal of supporting open models of production of services, as well as the possibility of reusing existing languages, tools and engines for developing a common infrastructure for pervasive service ecosystems. Candidate frameworks to reach this goal include the suite of standards of the Semantic Web [26, 40, 42].

2.2 Abstract architecture

Having introduced the scenario and its requirements, we now define the set of core components that constitute our pervasive ecosystem architecture.

Agents Any software component whose services act to make the pervasive system working is here modelled as an agent. It can be for instance a sensor, a web-service, the software handling a user profile into a smartphone, a situation-recogniser, a display driver or a full-fledged application.

LSA Because of the need of coordinating different kinds of entities in an open way and without global supervision, a cornerstone of pervasive ecosystems is that a *uniform representation* is required for agents, exposing any information about the agent (state, interface, goal, knowledge) that is pertinent for the ecosystem as a whole or for any subpart of it. This is called a "Live Semantic Annotation" for it should continuously represent the state of its associated component (live), and it should be implicitly or explicitly connected to the context in which such information is produced, interpreted and manipulated (semantic)—relying on standard languages for the description of resources, like RDF [26, 40].

LSA-space To handle situation-awareness, the behaviour of each agent should be strictly affected by the local context in which it runs, that is, on the state of other agents living in the same *locality* (intended as network neighbourhood). As such, the LSAs of each agent are reified in a distributed space (called an "LSA-space") acting as the *fabric* of the ecosystem, where "context" is simply defined and represented as the set of LSAs stored in a given locality.

LSA bonding Additionally, and in order to make any agent act in a meaningful way with respect to the context in which it is situated, special mechanisms are needed to provide a fine-tuned control of what is visible to/modifiable by each agent and what is not. We tackle this issue by allowing an LSA to include *bonds* (i.e., references) to other LSAs in the same context. It is only via a bond that an agent can inspect the state/interface of another agent and act accordingly, while modifications are allowed only to the LSAs an agent injected itself.

Eco-laws Because of adaptivity, while agents enact their *individual* behaviour by observing their context and updating their LSAs, *global* behaviour (i.e., global system coordination) is enacted by manipulation rules of the LSA-space, called *eco-laws*. They can execute deletion/update/movement/re-bonding actions applied to a small set of LSAs in the same locality. They are structured as chemical-resembling reactions over LSAs, similarly to other approaches like [4, 34, 36]—and their definition should leverage standard languages for the manipulation of resource descriptions, like SPARQL [1, 40].

Figure 1 shows an architectural view, based on the above abstractions, of a portion of an ecosystem featuring: two smartphones (carried by people) and two public displays forming a network of 4 computational nodes; a local LSA-space and some agents running in each node (e.g., recommendation agents, advertising agents, visualisation agents in displays, profile agents and sensor agents in smartphones); LSAs through

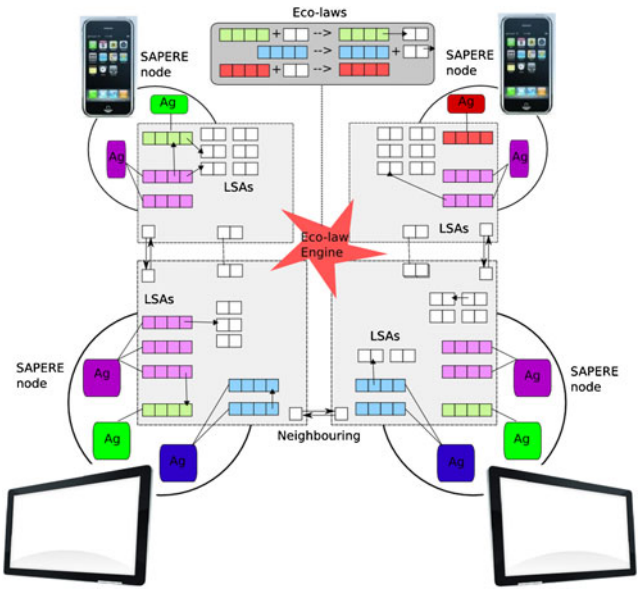

Fig. 1 An architectural view of a pervasive ecosystem

which agents manifest (in colour); additional LSAs representing data, knowledge, and contextual information like the existence of neighbouring nodes (in white); bonds between LSAs; and a set of eco-laws executed by an underlying engine working over the global LSA-space.

In a more general case, one should think at a very larger and mobile set of devices connected to each other based on proximity creating a distributed "space"—ideally a pervasive continuum—where LSAs form spatial structures evolved over time.

2.3 Operational model

We now describe in more detail the model of LSAs and eco-laws, focussing on the dynamics of their interaction and grounding it on standard frameworks and technologies for the Semantic Web, due to their support for openness (supporting interactions with third party software and data) and semantic reasoning (relying on ontologies and semantic matching) [40]. As already mentioned, we shall use RDF as language for structuring LSAs, and rely on SPARQL/SPARUL query languages for coding eco-laws: the main advantage of this choice is that off-the-shelf query engines (supporting execution of SPARQL queries and updates over RDF stores) and reasoners [29] can be used to support scheduling and execution of eco-laws locally.

2.3.1 LSAs

LSAs have a unique, system-wide identifier (LSA-id), needed to support a notion of identity that is key both to uniquely identify the agents that injected an LSA and to properly support a bonding mechanism based on reference rather than on value/copy. We refer to the content of an LSA as its *description*, which includes all the information the agent wants to manifest to the ecosystem. We realise an LSA as an RDF-like [26] set of triples that consist of a subject (an LSA-id), a predicate (the property name) and an object (the assigned value). By adopting a notation resembling N3 [18], in which the list of triples is provided in a more compact form, an LSA is represented, for example, as "`id p v; q w1 w2 w3.`" where `id` is the LSA-id, property `p` is assigned to value `v`, and property `q` is assigned to values `w1`, `w2`, and `w3`. Concretely, each element of a triple is an URI (a term qualified by a universally-accessible namespace as in `namespace:term`); additionally, a value can be an RDF literal (a string), or a description included into square brackets, recursively having the form of an LSA without identifier and trailing dot, e.g., "`[p v; q w1 w2 w3]`". Concrete examples of LSAs will be given in Section 5 (e.g. in Fig. 4).

2.3.2 Contextualisation and LSAs

In order to support situation-awareness, LSAs should be contextual, i.e., carry some information about their current context, and/or the context in which they have been created. Accordingly, either implicitly (enacted by the middleware) or explicitly (coded by the agent creating it) an LSA's semantic description includes, among the others, information like: current location, location of creation, creation time, last update time, creator, and so on. We call these *synthetic properties*: e.g., we shall use synthetic property `eco:location` to hold the id of the node in which the LSA is currently stored.

Contextual information not related to any specific agent is reified as a new LSA (which we call a *synthetic LSA*), which some middleware component is in charge of creating and updating. These LSAs contain information concerning the physical situation, such as current time, neighbouring nodes, and so on. In each node, we shall assume that for each neighbouring node there is one LSA of type `eco:neighbour` holding information such as orientation and estimated distance to it, and a single LSA of type `eco:time` holding information about the local time—the rate at which

they are actually updated, or whether they are updated only when accessed, is not prescribed by our model.

2.3.3 Eco-laws

Eco-laws are structured as chemical-resembling rules [4] with the syntactic structure:

```
P+..+P --r--> Q+..+Q SideConditions
```

Elements `P` and `Q` are patterns of LSAs, expressed like LSAs but with the following changes: (i) in place of each element of a triple one can use a variable `?V`; (ii) constraints on such variables are specified into an unordered sequence of side conditions, which are either "`FILTER(`*exp*`)`" or "`BIND(`*exp* `as ?V)`" (following the syntax of SPARQL FILTER and BIND constructs [1]); (iii) each predicate in a triple can be prepended by either symbol +, − and =, the former assumed by default—respectively meaning that the triples with this object should exist, should not exist, should be the only that exists for that subject and predicate. Note that we may use external functions to add computation abilities to the eco-law language, by using them into *exp* expressions of FILTER and BIND constructs.

An eco-law consumes a set of *reactant LSAs* based on left-hand side patterns and produce a set of *product LSAs* based on right-hand side patterns. It also obeys a numeric transformation rate `r` representing a Markovian rate in a continuous-time Markov chain (CTMC) system—though the underlying infrastructure may rely on approximations of this stochastic model for efficiency purposes. Rate `eco:asap` is used for eco-laws to be executed with "as soon as possible" semantics, namely, with infinite rate—namespace `eco` will be used for all concepts related to the pervasive ecosystems in general. An eco-law can apply in many different locations of the ecosystem, and to different sets of LSAs. We call *reaction* the pair consisting of a set of reactant LSAs and corresponding product LSAs that an eco-law can trigger. Execution of a reaction amounts to atomically remove reactant LSAs from the LSA-space and insert product LSAs back. Among all reactions that an eco-law can trigger, a node *n* schedules only those that are *effective* (reactant and product LSAs do not coincide, i.e., execution has a neat effect), *consistent* (do not invalidate uniqueness of LSA-ids), and *local* (reactants are located in *n*, products possibly in *n*'s neighbourhood—so that a reaction can also ultimately move or diffuse some LSA from its current location to a neighbouring one, and by repeated application, to larger regions).

3 Self-organisation for pervasive systems

Several recent works exploit the lessons of adaptive self-organising natural systems to enforce situation-awareness and adaptation in distributed and pervasive computing systems [3, 16, 27]. In natural systems (at the physical, chemical, biological, or social level), all the activities of the system components are inherently situated in space and driven by local interactions only. Such interactions are not ruled by pre-defined orchestrated patterns. Rather, interactions are simply subject to a limited set of natural laws, from which even complex patterns of interactions dynamically emerge via self-organisation. In this way, adaptivity becomes an inherent characteristic deriving from the existence of self-organising interactions patterns, whose structure can flexibly and robustly re-shape in response to contingencies. Typical self-organising mechanisms are those using stigmergy, like ant foraging for coordinating behaviour, schooling and flocking for coordinating movements, or gradients based systems [7, 10, 28, 41].

3.1 Self-organising patterns

To make self-organising mechanisms applicable more systematically, different authors have focussed on proposing descriptions of those mechanisms under the form of software design patterns [13] and their classification. The idea of design pattern structure makes it easy to identify the *problems* that each mechanism can solve, the specific *solution* that it brings, the *dynamics* among the entities and the *implementation*. In [14], a set of design patterns is proposed for self-organising systems all related with ant colonies behaviour, together with the idea that a mechanism can be composed with others. The provided model, however, presents too many constraints to be generalised and the examples of usage are not related to spatial and situated computing systems as typically required. Based on the set of mechanisms proposed in [17], [31] discusses how the intended multi-agent systems (MAS) dynamics can be modelled and refined to decentralised MAS design, proposing a systematic design procedure that is exemplified in a case study. In [8] it is presented an extended catalogue of mechanisms as design patterns for self-organising emergent applications. The patterns are presented in detail and can be systematically applied for engineering self-organising systems. However, relations among the patterns are missed, i.e., the authors do not describe how patterns can be combined to create new patterns or adapted to tackle different problems.

An approach more useful here was presented in [11], which we shall adopt in this paper, where a set of

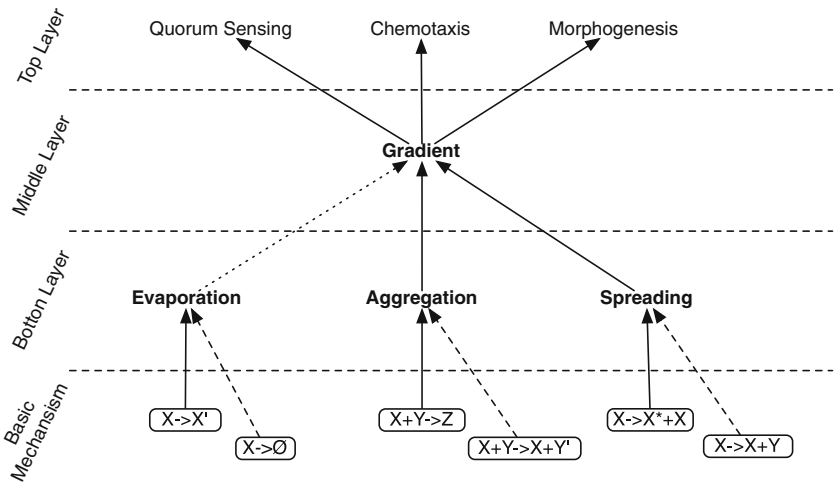

Fig. 2 Self-organising patterns, their relationships, and their constituting mechanisms

bio-inspired self-organising mechanisms are analysed, classified and described, identifying their relations and the recurrent problem they solve—see Fig. 2 including a fragment of that catalogue. The result that comes out from there is that most natural-inspired algorithms presented in the literature can be designed and implemented as extensions and compositions of low level patterns of Spreading, Aggregation and Evaporation. The spreading pattern allows entities to increment the global knowledge of a system by periodically sending information from one entity to another. To avoid an explosion of information in the system, the aggregation pattern synthesises this information, extracting meaningful information. Finally, evaporation decreases information's relevance over time (i.e., making information deposited recently more relevant than information deposited previously). Moreover, in [11], it is suggested that these low-level patterns can be modelled by simple transformation rules, paving the way towards design and implementation in general-purpose platforms—as we develop in this paper.

3.2 Execution models for self-organising patterns

A previous proposal for supporting the patterns of [11] into a general-purpose abstract architecture inspired to the pervasive service ecosystems framework is presented in [12]. Such an abstract execution model is presented providing low-level functionalities for designing and implementing self-organising systems. The corresponding computational model includes: (i) agents—autonomous and pro-active software entities running in a host; (ii) infrastructure—a set of connected hosts and infrastructural agents; (iii) environment—the physical space where the infrastructure is located. It is then composed of: (i) a core's Data Space, where agents deposit and retrieve data; (ii) a set of basic bio-inspired services implementing low-level patterns through rules applying on data deposited in the data space; and (iii) core interfaces providing primitives for the infrastructural agents to access neighbouring nodes, and sensors and actuators of the local node.

In this paper we follow this idea, and develop it to full extent in the context of the pervasive service ecosystems framework as described in Section 2.

4 Self-organisation for pervasive ecosystems: a model

Following the above described approaches, we here show how self-organising patterns can be mapped into the framework presented in Section 2.3. Accordingly, we will identify (i) a set of basic chemical-like rules in the form of chemical reactions (Table 1), (ii) the constituting relations of these basic rules with low-level patterns (Fig. 2), and (iii) a model for basic patterns in terms of LSAs and eco-laws whose details are given in the next section.

Table 1 Set of basic rules

Bio-chemical reaction	Name
$X \rightarrow X'$	Evolution
$X \rightarrow \emptyset$	Decay
$X \rightarrow X^* + X$	Diffusion
$X + Y \rightarrow X + Y'$	Contextualisation
$X + Y \rightarrow Z$	Composition
$X \rightarrow X + Y$	Synthesis

The chemical representation is rather standard—we marked by a "*" those LSAs whose locations will be changed by the reaction, namely, which will be spread in neighbours

4.1 A set of basic rules and eco-laws

Relying on the ecosystem model, we equip each LSA-space with a minimal set of eco-laws that model chemical-like reactions as in Table 1. We shall demonstrate that by a simple composition of these basic rules, Spreading, Aggregation and Evaporation autonomously emerge as low-level patterns for providing thereby systems with higher level ones, such as Gradient. Note that the set of basic rules that we identified is not intended to be complete with respect to real-world phenomena, but sufficient for supporting the low-level patterns of Fig. 2. The representation of these rules in terms of eco-laws is formalised in Fig. 3.

In particular, LSAs that once injected are meant to be subject to one of such eco-laws feature a non-empty property `sos:request`—namespace `sos` will be used for all concepts related to the support of self-organisation. Such a property has, as values, descriptions that define all the elements that are necessary to identify which eco-law should apply and its parameters, such as rate, operators/predicates (in the form of URIs or strings, as usual), and the property that is going to be evaluated and possibly changed by the eco-law execution. Hence, the `sos:request` property can specify more descriptions, if the LSA is expected to be part of more then one eco-law.

Consider eco-law *[EVOLUTION]*. It expresses the fact that if an LSA with id `?LSA` (namely, whose id gets binded to variable `?LSA`) features a `sos:request` for evolving the content of property `?P` by operator `?E_op` with rate `?R`, and it has property `?P` assigned to value `?C`, then the eco-law will actually fire with rate `?R`. Its effect would be to update property `?P`, which will be assigned to the result of applying `?E_op` to `?C`—this is achieved by a BIND side-condition, in which library function `eco:exec` is used to apply operator `?E_op` to value `?C`.

Note that `?E_op` will have to bind to an URI or string that the eco-law engine in the middleware can interpret as a function over values (primitive ones or descriptions). As an example, by a request of the kind:

```
[ev:rate "1.0"; ev:prop ex:p;
 ev:op ex:dividebytwo]
```

Eco-laws for Basic Mechanisms

```
[EVOLUTION]: An LSA evolves its content
?LSA sos:request [ev:rate ?R; ev:prop ?P; ev:op ?E_op]; ?P =?C
--?R-->
?LSA ?P =?D
BIND (eco:exec(?E_op,?C) AS ?D )

[DECAY]: An LSA evolves its content
?LSA sos:request [dec:rate ?R; dec:prop ?P; dec:predicate ?D_pr]; ?P =?C
--?R-->
0
FILTER (eco:check(?D_pr,?C))

[DIFFUSION]: An LSA diffuses a version in a neighbour
?DIF sos:request [df:rate ?R; df:prop ?P; df:op ?D_op]; eco:location ?L; ?P =?C +
?NG eco:type eco:neighbour; eco:neigh_location ?L1; eco:distance ?D
--?R-->
?DIF + ?NG + ?NEI eco:location = ?L1; ?P = ?E; eco:prev =?L
BIND (eco:clone(?DIF) AS ?NEI)
BIND (eco:exec3(?D_op,?C,?D) AS ?E)

[CONTEXTUALISATION]: An LSA contextualises to its environment
?LSA sos:request [ctx:rate ?R; ctx:prop ?P; ctx:ctx_prop ?P2; ctx:op ?C_op]; ?P =?C + ?CTX ?P2 =?C2
--?R-->
?LSA ?P =?D + ?CTX
BIND (eco:exec3(?C_op,?C,?C2) AS ?D)

[COMPOSITION]: An LSA composes with an other one
?LSA sos:request [cmp:rate ?R; cmp:prop ?P; cmp:cmp_prop ?P2; cmp:op ?C_op]; ?P =?C + ?COMP ?P2 =?C2
--?R-->
?LSA2 ?P =?D
BIND (eco:clone(?LSA) AS ?LSA2)
BIND (eco:exec3(?C_op,?C,?C2) AS ?D)

[SYNTHESIS]: An LSA is synthesised
?LSA sos:request [syn:rate ?R; syn:prop ?P ; syn:syn_prop ?P2] + ?TIME eco:time ?T
--?R-->
?LSA + ?TIME + ?SYN sos:syn_time =?T
BIND (eco:clone(?LSA) AS ?SYN)
FILTER (eco:cloneprop(?LSA,?P,?SYN,?P2))
```

Fig. 3 Eco-laws for the basic rules in Table 1

where ex:dividebytwo represents the functions that divides a real number by two, we make the LSA halving its property ex:p once per time unit—we typically use seconds as time units. We shall assume that inside the middleware (e.g., enconded in the ex ontology), URI ex:dividebytwo is associated with a function whose evaluation provides the required result. To this end, in this paper, we describe also such functions by mathematical expressions, and denote arguments of the function orderly as #1, #2 and so on, such that ex:dividebytwo would be described also by string expression "%1/2"—other notations will be explained as needed in the following. Also, when application of an operator yields the special value eco:error, then the eco-law is not triggered at all for that LSA, in that function eco:exec will fail.

The *[DECAY]* eco-law is similar, but it makes ?LSA disappear (note the 0 on the right-hand side) at rate ?R if the content ?C of its property ?P is such that predicate ?D_pr holds for it—function eco:check in construct FILTER is used to perform this test. E.g., we could make an LSA be disposed if a certain property ex:p reaches (for instance because of the above *[EVOLUTION]* eco-law) a value smaller than 0.01—the content of dec:prop is to be set to ex:p and dec:predicate to expression "#1<0.01" to this end.

The other eco-laws are a bit more involved for they manipulate more than one LSA, but their behaviour is similar. Eco-law *[DIFFUSION]* continuously diffuses an LSA in neighbouring locations, one at a time. Let ?NG be a synthetic LSA reifying information about the existence of a neighbouring node at location ?L1 and estimated distance ?D. Also, let ?DIF be an LSA which, at rate ?R, is aimed at diffusing clones of it with an updated value of property ?P, namely, obtained by applying operator ?D_op. Then, at rate ?R we create a new LSA ?NEI (?DIF and ?NG are not changed), obtained by cloning ?DIF (using function eco:clone in the BIND construct), locating it at ?L1, and changing property ?P to the result of applying ?D_op to current value ?C and to the estimated distance ?D—we use value ?D since often diffusion depends on the distance from the chosen neighbour, and rely on function eco:exec3 as a ternary version of eco:exec. By iterative application, this eco-law is used to diffuse copies of ?DIF in all neighbours.

By eco-law *[CONTEXTUALISATION]*, LSA ?LSA is updated by the presence of an LSA ?CTX nearby: the content of its ?P property is updated by the result of applying ?C_op to the old value ?C and to the current value of property ?P2 in ?CTX. The *[COMPOSITION]* eco-law works similarly, but the result of its application is that the two originating LSAs are removed, and a new one cloning ?LSA is created with an update value of ?P. Finally the *[SYNTHESIS]* eco-law allows for the creation of a new LSA cloning an existing one (?LSA) featuring a new property ?P2 assigned to the content of ?LSA's ?P, and property synth_time assigned to the current time ?T as extracted by the synthetic LSA ?TIME. In this case, function eco:cloneprop in the FILTER construct is used to transfer all the values assigned to a property into another LSA's property.

4.2 Basic patterns

This section is meant to describe how we intend to model low-level patterns upon the basic rules described. As shown in Fig. 2, constituting relations have been identified: arrows indicate how these patterns result from the composition of basic laws. A dashed arrow indicates that using the below eco-law is actual optional, i.e., the pattern itself can be realised also without that basic rule.

Spreading Pattern

> "The Spreading Pattern is a basic pattern for information diffusion/dissemination. The Spreading Pattern progressively sends information over the system using direct communication among agents, allowing the agents to increment the global knowledge of the system by using only local interactions" [12].

Therefore Spreading mainly relies upon the Diffusion rule, but it can also require the Synthesis rule when the diffusing data has to be synthesised by an original LSA. The Synthesis rule is a specific instance of the one shown in Fig. 3, where the synthesised LSA has to contain the sos:request property value enabling the diffusion.

Aggregation Pattern

> "The Aggregation Pattern, is a low-level pattern for information fusion. The dissemination of information in large-scale systems deposited by the agents or taken from the environment may produce network and memory overload, thus, the necessity of synthesising the information. The Aggregation Pattern reduces the amount of information in the system and assesses meaningful information" [12].

In our model the Aggregation Pattern firstly results from the application of the Composition rule where two data in input are aggregated into a new information through an operator that can take many forms, such

as filtering, merging, transforming. This rule can also be composed with the Contextualisation rule, by which aggregation can be affected also by some contextual LSA present in the LSA-space but which is not to be changed or removed. These two basic rules provide a minimal set of functions for the Aggregation Pattern working. The idea is that they are fired repetitively (and typically with a very high rate) until aggregation can no longer be applied, namely when it leads to an atomic information.

Evaporation Pattern

> "Evaporation is a pattern that helps to deal with dynamic environments where information used by agents can become outdated. In real world scenarios, the information changes with time and its detection, prediction, or removal is usually costly or even impossible. Thus, when agents have to adapt their behaviour according to information from the environment, information gathered recently must be more relevant than information gathered a long time ago" [12].

In our model the Evaporation Pattern is mainly obtained through the Evolution rule, that ensures the reduction of relevance of information through a proper operator. It can be composed with the Decay rule once the relevance of the information is null or very low and must be disposed of.

Gradient Pattern The Gradient Pattern is an example of computational field: a data-structure distributed in a networked system based on spatial abstractions (distance, region, paths, and so on). Computational fields are a remarkable self-organisation mechanism, as demonstrated by their wide adoption in literature [5, 16, 23, 36]. The Gradient Pattern in particular maps each node to the minimum distance from a source. It is a very important pattern in pervasive computing systems, for it makes a possibly large set of nodes that surrounds a single one (the gradient source) be aware of its state (or parts of it), and aware also of how it can be reached efficiently (i.e., along the optimal path crossing nodes with decreasing distance)—hence supporting long-distance interactions in ad-hoc networks.

> "The Gradient Pattern focuses on large systems that suffer from lack of global knowledge to estimate the consequences of the actions performed by other agents beyond their communication range. Using the Gradient Pattern, information spreads from a location it is initially deposited and aggregates when it meets other information. Thus, agents that receive gradients have information that come from beyond their communication range, increasing the knowledge of the global system not only with gradient's information but also with the direction and distance of the information source" [12].

According to our model the Gradient Pattern emerges from the composition of the Spreading and Aggregation Patterns (as shown in Fig. 2), properly instantiated. In particular the Diffusion rule, in charge of diffusing data, has a specific operator that increases the distance value depending on the estimated distance of the neighbour, while the Composition and Contextualisation rules work respectively for merging gradient values coming from the same source but through different paths, so as to ensure that the smaller distance from the source is stored in each node, and for contextualising the gradient value to the actual node state, so as to advantage or penalise nodes/regions according to their actual context—as we will exemplify in detail in next section.

5 A crowd steering application

We here present the model for the crowd steering scenario described at the beginning of Section 2 to exemplify the approach and demonstrate how the eco-laws presented in previous section can be used to use and compose self-organisation patterns. In one recalling sentence, the goal of the scenario is to guide people inside a museum towards their preferences, following the shortest, and possibly quickest path, as can be dynamically computed from the structure of the environment and the presence of people making certain rooms or corridors too crowded.

We here model the whole ecosystem as the set of LSA-spaces hosted in the sensor nodes covering the museum floor, displaced in a grid-like manner. Nodes are connected to the four adjacent ones, following the structure of the environment. Steering of people can be done using computational gradients injected from sources of a point of interest (POI), diffusing around such that each node holds the minimum distance from source along an optimal path, and matching with some user preferences [5, 36]. We do not focus here on the details by which a user can decide to follow the gradient of a given POI among the many that can exist around—she can explicitly select one by interaction with the smartphone, or a match can be implicit. Once this has been selected, simply following the directions descending the gradient (as provided by private/public displays) leads to a proper path for the situation at

hand. If we want the gradient to be dynamically computed taking into account also the presence of crowd, the gradient value (estimated distance to the source) should be contextualised considering the presence of people, becoming higher in nodes detecting a bigger number of people around.

In the following we show how we are able to create such a contextualised gradient through the patterns and eco-laws presented in Section 4.1.

5.1 Specific LSAs

According to the proposed framework, all the information exchanged is encapsulated by LSAs, namely: (i - source LSAs) representing POIs currently active; (ii - field LSAs) representing diffused copies of source LSAs and carrying an updated estimated distance from the source along the best path available; (iii - pre-field LSAs) temporary copies of field LSAs, used as intermediate ones to enact aggregation and contextualisaton, and to establish the final gradient; (iv - user LSAs) representing presence and state of a user (stored in their smartphone); and (v - crowd LSAs) representing the presence of a crowded area by a sensor node. The shape of source LSAs is the most important to show here, for it is by its injection that field and pre-field LSAs get automatically created. The source LSAs shown in Fig. 4 exemplifies the POI of a Michelangelo's sculpture exhibition in a museum. Other than namespaces `eco` (for general concepts related to pervasive ecosystems) and `sos` (for those related to self-organisation patterns), we shall use `museum` for application-specific concepts. In a source LSA, `eco:type` keeps track of a general declaration about the kind of LSA, `eco:location` is the synthetic property (automatically generated by the middleware) holding the location id for the LSA, and `museum:poi_desc` holds a list of keywords describing the POI (used to match with a user preferences). The other properties will be described in the following.

The injection of a source LSAs fires the Synthesis basic mechanism because of the content of property `sos:request`, by which at rate 1 a new LSA is created locally, which—according to the [SYNTHESIS] eco-law—is similar to the source LSA but reassigns `sos:request` to the actual content of `museum:exh_request`. This new LSA represents the gradient at the source location: the new property `syn_time` records the time at which it has been generated, `museum:desc` holds the information to be propagated, and `museum:grad_state` the information about how one can retrieve the source. The latter is a description with a pair of a distance value (initially set to 0) and a pre-field flag tagging pre-fields—this is initially set to `false`, meaning this is not a pre-field, but a field LSA.

The resulting LSA includes now four requests into `sos:request`: one for diffusing, two for aggregating and one for contextualising, which we will describe in turn. Initially, diffusion rule spreads copies of that LSA around, updating the property `museum:grad_state` according to the `museum:diffsum` operator which has the following definition:

Definition of operator `museum:diffsum`

```
#1.museum:pre
 ? eco:error
 : [museum:distance #1.museum:distance + #2; "true"]
```

This function is applied to two arguments, the old `museum:grad_state` content, and the distance of the selected neighbour. Ternary operator `?:` has the same meaning of Java programs, while notation "`#1.museum:pre`" stands for the content of property `museum:pre` in the description passed as first argument. Hence, this function checks whether the property `museum:pre` into first argument is true: if it is, then we return an error since pre-fields are not to be diffused, otherwise we return a new description with updated distance (old distance plus neighbour distance) and setting it as a pre-field.

```
lsa:exhibition1432
   eco:type museum:exhibition;
   eco:location "node34165@room131";
   sos:request [syn:rate "1.0";  syn:prop museum:exh_request; syn:syn_prop sos:request];
   museum:exh_request
       [df:rate "10.0"; df:prop museum:grad_state; df:op museum:diffsum]
       [ctx:rate eco:asap; ctx:prop museum:grad_state; ctx:ctx_prop museum:crowd_level; ctx:op museum:crowd]
       [cmp:rate eco:asap; cmp:prop sos:syn_time; cmp:cmp_prop sos:syn_time; cmp:op museum:youngest]
       [cmp:rate "100.0"; cmp:prop museum:grad_state; cmp:cmp_prop museum:grad_state; cmp:op museum:shortest];
   museum:grad_state [museum:distance "0"; museum:pre "false"];
   museum:poi_desc "michelangelo" "david" "sculpture" "renaissance" .
```

Fig. 4 LSA for the museum case study

To a pre-field, shipped into neighbours by this eco-law, aggregation and contextualisation eco-laws can then be applied to remedy the inevitable divergence (an increasingly number of copies of a pre-field will be spread in neighbours). Eco-law for composition is fired which matches two LSAs and aggregate them into one in two ways: one keeps the most recent LSA (having greater `syn_time`), as computed by the operator `museum:youngest`, while the other keeps the one with smaller distance from the source, as computed by the operator `museum:shortest`. Such two operators are as follows:

Definition of operator `museum:youngest`

```
#1 < #2 ? sapere:error : #1
```

Definition of operator `museum:shortest`

```
#1.museum:distance > #2.museum:distance
   ? eco:error
   : [#1.museum:distance; "false"]
```

Note that the rates of application of aggregation with `youngest` is `eco:asap`, while aggregation with `shortest` has a finite rate: stochastically, this ensures that we first keep most recent information, and then we consider better paths.

Before such an aggregate LSA can be diffused again, it should contextualise with a crowd LSA, so as to make sure that the distance from the source gets increased (i.e., penalised) when the crowd level is locally greater than 0. Such an information is carried into a crowd LSA of the kind:

```
lsa:crowdlsa1123
    eco:type museum:crowd;
    eco:location "node34165@room131";
    eco:time "2011-05-30T11:00:00";
    museum:crowd_level "1.0" .
```

where `museum:crowd_level` set to 1.0 means that the sensor perceived the highest crowd. The eco-law for contextualisation is then triggered which takes a pre-field LSA and a crowd LSA, and updates property `museum:distance` in the former according to the function:

Definition of operator `museum:crowd`

```
#1.museum:pre
  ? [#1.museum:distance + #2 * museum:crowd_factor;
     "false"]
  : eco:error
```

`museum:crowd_factor` is a multiplication factor dictating how `museum:crowd_level` should penalise estimated distance.

5.2 Simulation

By the structure of the LSAs and eco-laws described above, we are able to maintain and contextualise a stable gradient. As a proof-of-concept for the proposed solution, we rely on simulations of the evolution of the population of LSAs. As such, once the initial state of LSAs and eco-laws are fixed, the evolution of a service ecosystem can be simulated using any available framework for CTMCs, typically working via Stochastic Simulation Algorithms (SSA) based on [15].

We performed simulations conducted over an exposition of nine rooms connected via corridors. A first set of tests was aimed at testing the effectiveness of gradients in the process of steering to a destination, even in an averagely-crowded situation. Four snapshots of a simulation run are reported in Fig. 5, where we considered four different targets located in the four rooms near environment edges. People (each having interest in one of the targets chosen randomly) are initially spread randomly in the museum, as shown in the first snapshot, and they eventually reach the room in which the desired target is hosted, as shown in the last snapshot.

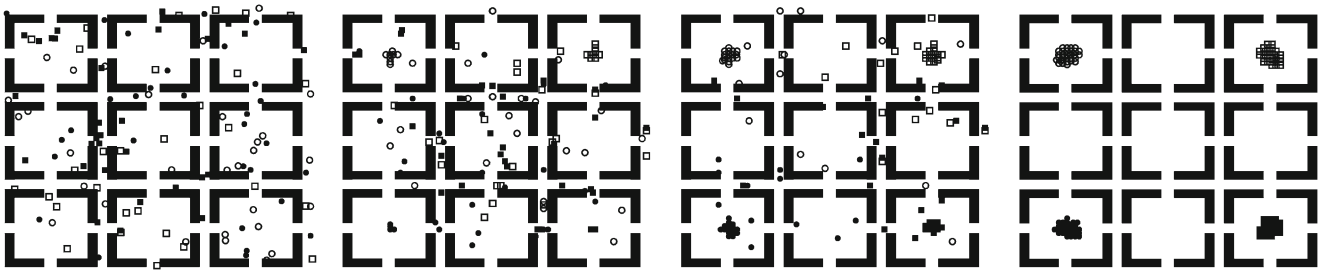

Fig. 5 A simulation run of the reference exposition (*top–left*, *top–right*, *bottom–left*, *bottom–right*): from random positions people move to 4 targets

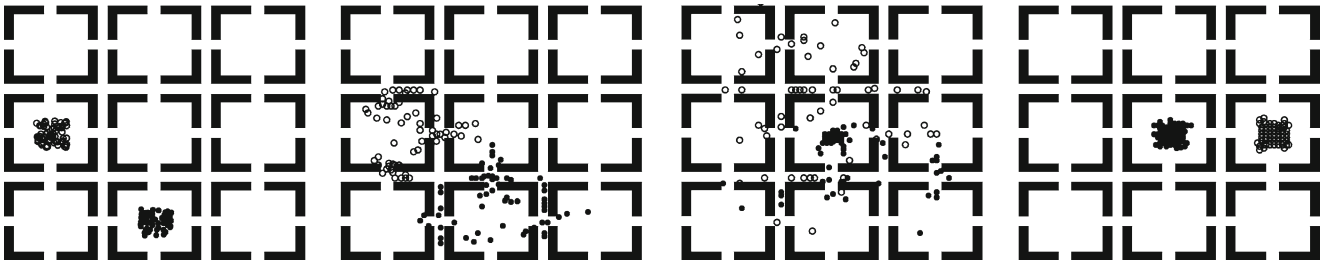

Fig. 6 Dark visitors occupy a central room: others move *left to right* by a longer, less crowded path circumventing the central room on *top*

A second set of tests was aimed at verifying the management of overcrowding, and in particular, how the behaviour of the ecosystem can dynamically and automatically become self-aware of crowding conditions, and react accordingly. Figure 6 shows another simulation run: two groups of people, each with a common interest in an exhibition—denoted with empty (light) and filled (dark) circles—are initially located in two different rooms, as shown in the first snapshot. The target for the dark visitors is located in the central room of the second row, while the others' is in the right room of the second row. In the simulation, dark visitors reach their target first because it is closer, however, the resultant crowded area formed intersects the shortest path towards the other visitors' target. Due to this jam the latter visitors are guided along a different path, which is longer but less crowded. Also note that people do not all follow the same path to a destination, but rather spread and take several different paths to the POI by an emergent "self-crowding" phenomenon: people dynamically tend to follow different paths to avoid themselves to make some corridor or room too crowded.

Both tests show qualitative effectiveness of the proposed eco-laws, and suggest that our simulation approach can be used for additional experiments focussing on tuning system parameters (crowd factor k) or alternative strategies (e.g., diffusing crowd information) to optimise paths to destinations. For instance, in the context of the second case, Fig. 7 shows how factor k can influence the time for (sub)groups of (light) people to reach the destination, by which we can see that even small values of k lead to a significant improvement—which slowly decreases as k grows.

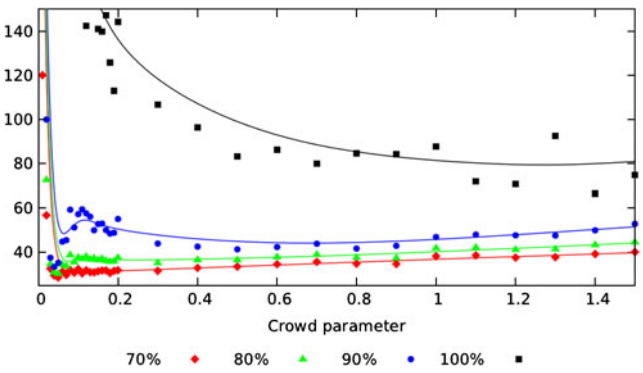

Fig. 7 Time units of convergence time with different values of crowd parameter and different percentages of people

6 Related work

We already surveyed the existing approaches to self-organisation patterns in Section 3.1. In this section we review the existing coordination models and middlewares that could be used to support those patterns, as a possible alternative to the ecosystem model presented here as an extension to [37]. Most of them are based on tuple space architectures, which we describe in terms of how they support basic mechanisms of diffusion and management of tuples.

As described in [19], applications of coordination models and languages—and especially space-based ones—are inevitably entering the realm of self-organisation, where complexity of interactions becomes the key to make desired properties appear by emergence. Given the intrinsic difficulty of *designing emergence*, most approaches simply mimic nature-inspired techniques to organise and evolve tuples according to specified rules. Anthill [2] is a framework built to support design and development of adaptive peer-to-peer applications. It consists of a dynamic network of peer nodes, each one provided with a local tuple space, in which distributed mobile agents can travel and can indirectly interact and cooperate with each other by leaving and retrieving tuples. Self-organisation in Anthill is realised by agents, without additional mechanisms "in the space". A similar idea is applied in [38], in which tuples can create spatial processes defining evolving regions to be used for coordination in mobile networks. SwarmLinda [32] is a

middleware that exploits the idea of the collective intelligence displayed by swarms of ants for guiding agents in charge of tuple storage and efficient tuple retrieval. Tuples are handled as sort of pheromones or items that ants (agents) relocate in order to improve overall efficiency. TOTA (Tuples On The Air) [16] is a tuple-based middleware supporting field-based coordination for pervasive-computing applications. In TOTA each tuple, when inserted into a node of the network, is equipped with a content (the tuple data), a diffusion rule (the policy by which the tuple has to be cloned and diffused around) and a maintenance rule (the policy whereby the tuple should evolve due to events or time elapsing). The *evolving tuples* model, presented in [30], is an extension to traditional Linda tuple spaces with the goal of supporting resource discovery in a pervasive system, relying on ideas inspired to TOTA. The extension allows tuples to evolve so to be context-aware and able to adapt to environmental changes. Evolution is firstly embedded in tuples by adding, to each field of the tuple, a name and a formula that specifies the field behaviour over time. Formulas support if-then-else construct and arithmetic and boolean operators. Secondly a new operation `evolve()` is introduced in tuple space: it is responsible for applying formulas to tuples using context information.

Finally, it is worth noting that the pervasive ecosystem model originates from previous work of ours [34, 36], where a bio-chemical tuple spaces model has been presented. There, tuples are associated with an *activity level*, which resembles chemical concentration and measures the extent to which the tuple can influence the state of system coordination—e.g., a tuple with low activity level would be rather inert, hence taking part in coordination with very low frequency. Chemical-like reactions, properly installed into the tuple space, evolve activity level of tuples over time in the same way chemical concentration is evolved in chemical systems.

Differences of the above models with that of pervasive ecosystems is as follows. Behaviours that allow system evolution and adaptation are not embedded in tuples but in space so to guarantee multiple behaviours in different locations of the network. Moreover our model introduces probability in the selection of which rule to execute, enabling the reproduction of a wider range of mechanisms for modelling system evolution. More in detail, and concerning [30], our model lets two or more tuples be part of an eco-law that changes one or more tuples according to the state of the others, while in [30] each tuple evolves independently from the others. Concerning TOTA, which is the model more similar to ours also from the viewpoint of application domain, we observe that while eco-laws are meant to be fixed for the application domain and apply to all LSAs (depending on semantic matching criteria), in TOTA each tuple is responsible for carrying its behavioural rules. So, while we call for specifying the evolution rules of tuples at design-time, when the application goals are identified, TOTA instead promotes a run-time approach: diffusion behaviour is defined by an agent before injecting the tuple in the system. Embedding the behaviour in the tuples, rather than in the space, makes difficult and basically impractical the task of predicting overall ecosystem behaviour in advance.

Finally, we note that the approach presented in this paper might be connected with a more general view of agent-based systems, provided we rely on agent meta-models tackling the environment as a first-class notion, as developed in [20, 21].

7 Conclusion and future work

To support situation-awareness and adaptivity in pervasive computing applications, in this paper we propose to exploit the lesson of self-organisation in natural systems, that—by grounding solely on distributed, locally-scoped interactions—autonomously brings self-* properties in the whole system. Recent works identified a set of self-organising patterns and classified them into a hierarchy where patterns at lower level represent the constituting behaviours for self-organisation to emerge. To model these patterns and show how they can be composed to obtain higher level patterns, we adopt a new framework of pervasive ecosystems based on the eco-law and LSA abstractions—the former regulates the overall ecosystem by basic mechanisms of spatial coordination; the latter regulates agent autonomy, controlling how the single agent is affected by the ecosystem and manifests to it. In particular we modelled eco-laws in terms of basic chemical-like mechanisms of diffusion, decay, composition and contextualisation. An example application in the context of pervasive computing is provided to clarify the concepts. Simulations of the associated CTMC semantics have been conducted to qualitatively and quantitatively validate the resulting behaviour.

Future works of this research include: (i) studying connection with Web technologies for representing and reasoning about state and behaviour, as well as to support information matching as in the Semantic Web; (ii) integrate in the system external software entities that perform more elaborated forms of situation recognition, relying on self-organising patterns for shaping data within the network of pervasive devices; (iii) investigate upon use cases of pervasive computing other design

patterns with the goal of identifying a minimal set of design patterns required for supporting such applications; (iv) studying analysis techniques, such the ones from complex system sciences, to predict and control the system emergent behaviour such that to avoid undesirable effects; and (v) develop a prototype infrastructure to support LSA-spaces.

Acknowledgements This work has been supported by the EU-FP7-FET Proactive project SAPERE Self-aware Pervasive Service Ecosystems, under contract no.256873.

References

1. ARQ (2012) A SPARQL processor for Jena. http://jena.sourceforge.net/ARQ/
2. Babaoglu O, Meling H, Montresor A (2002) Anthill: a framework for the development of agent-based peer-to-peer systems. In: Proceedings of the 22nd international conference on distributed computing systems (ICDCS'02), ICDCS '02. IEEE Computer Society, Washington, DC, USA, pp 15–22. http://dl.acm.org/citation.cfm?id=850928.851860
3. Babaoglu O, Canright G, Deutsch A, Caro GAD., Ducatelle F, Gambardella LM, Ganguly N, Jelasity M, Montemanni R, Montresor A, Urnes T (2006) Design patterns from biology for distributed computing. ACM Trans Auton Adapt Syst 1(1):26–66. doi:10.1145/1152934.1152937
4. Banâtre JP, Priol T (2009) Chemical programming of future service-oriented architectures. J Softw 4(7):738–746
5. Beal J, Bachrach J (2006) Infrastructure for engineered emergence on sensor/actuator networks. IEEE Intell Syst 21(2):10–19. doi:10.1109/MIS.2006.29
6. Bonâtre JP, Le Métayer D (1996) Gamma and the chemical reaction model: ten years after. In: Coordination programming. Imperial College Press London, UK, pp 3–41
7. de Castro LN (2006) Fundamentals of natural computing: basic concepts, algorithms, and applications (Chapman & Hall/Crc Computer and Information Sciences). Chapman & Hall/CRC
8. De Wolf T, Holvoet T (2007) Design patterns for decentralised coordination in self-organising emergent systems. In 4th international workshop, ESOA 2006. Hakodate, Japan, May 9, 2006. LNCS, vol. 4335. Springer, pp 28–49
9. Di Pierro A, Hankin C (2005) Wiklicky H continuous-time probabilistic KLAIM. ENTCS 128(5):27–38
10. Dressler F, Akan OB (2010) A survey on bio-inspired networking. Comput Networks 6:881–900
11. Fernandez-Marquez J, Di Marzo Serugendo G, Montagna S, Viroli M, Arcos J (2012) Description and composition of bio-inspired design patterns: a complete overview. Nat Comput 1–25. doi:10.1007/s11047-012-9324-y
12. Fernandez-Marquez JL, Serugendo GDM, Montagna S (2012) BIO-CORE: bio-inspired self-organising mechanisms core. In: Hart E, Timmis J, Mitchell P, Nakamo T, Dabiri F, Akan O, Bellavista P, Cao J, Dressler F, Ferrari D, Gerla M, Kobayashi H, Palazzo S, Sahni S, Shen XS, Stan M, Xiaohua J, Zomaya A, Coulson G (eds) Bio-inspired models of networks, information, and computing systems. Lecture notes of the institute for computer sciences, social informatics and telecommunications engineering, vol 103. Springer, Berlin Heidelberg, pp 59–72. doi:10.1007/978-3-642-32711-7_5
13. Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns: elements of reusable object-oriented software. Addison-Wesley, Reading, Mass
14. Gardelli L, Viroli M, Omicini A (2007) Design patterns for self-organising systems. In: Burkhard HD, Verbrugge R, Varga LZ (eds) Multi-agent systems and applications V. LNAI, vol 4696. Proceedings 5th international central and eastern European conference on multi-agent systems (CEEMAS'07), Leipzig, Germany, 25–27 Sep 2007. Springer, pp 123–132
15. Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81(25):2340–2361
16. Mamei M, Zambonelli F (2009) Programming pervasive and mobile computing applications: the tota approach. ACM Trans Softw Eng Methodol 18(4):1–56. doi:10.1145/1538942.1538945
17. Mamei M, Menezes R, Tolksdorf R, Zambonelli F (2006) Case studies for self-organization in computer science. J Syst Archit 52:433–460
18. Notation3 (n3) (2011) A readable rdf syntax. http://www.w3.org/TeamSubmission/n3/
19. Omicini A, Viroli M (2011) Coordination models and languages: from parallel computing to self-organisation. Knowl Eng Rev 26(1):53–59. doi:10.1017/S026988891000041X. Special issue 01 (25th Anniversary issue)
20. Omicini A, Ricci A, Viroli M (2006) Coordination artifacts as first-class abstractions for MAS engineering: state of the research. In: Garcia AF, Choren R, Lucena C, Giorgini P, Holvoet T, Romanovsky A, (eds) Software engineering for multi-agent systems IV: research issues and practical applications. LNAI, vol 3914. Springer, pp 71–90. doi:10.1007/11738817_5. http://www.springerlink.com/link.asp?id=t710627571v4256h (Invited Paper)
21. Omicini A, Ricci A, Viroli M (2008) Artifacts in the A&A meta-model for multi-agent systems. Auton Agent Multi-Ag 17(3). doi:10.1007/s10458-008-9053-x. http://www.springerlink.com/content/l2051h377k2plk07/
22. Paun G (2002) Membrane computing: an introduction. Springer-Verlag New York, Inc., Secaucus, NJ, USA
23. Pianini D, Montagna S, Viroli M (2011) A chemical inspired simulation framework for pervasive services ecosystems. In: Ganzha M, Maciaszek L, Paprzycki M (eds) In: Proceedings of the federated conference on computer science and information systems. IEEE Computer Society Press, Szczecin, Poland, pp 675–682
24. Picco GP, Murphy AL, Roman GC (1999) LIME: Linda meets mobility. In: The 1999 international conference on software engineering (ICSE'99), May 16–22, Los Angeles (CA), USA. ACM, pp 368–377
25. Priami C (1995) Stochastic pi-calculus. Comput J 38(7):578–589
26. RDF primer (2012) http://www.w3.org/TR/rdf-primer/
27. Ricci A, Omicini A, Viroli M, Gardell, L, Oliva E (2007) Cognitive stigmergy: towards a framework based on agents and artifacts. In: Weyns D, Parunak HVD, Michel F (eds) Environments for multiagent systems. LNAI, vol 4389. 3rd international workshop (E4MAS 2006), Hakodate, Japan, 8 May 2006. Springer, pp 124–140 (Selected revised and invited papers)
28. Serugendo G, Gleizes M, Karageorgos A (2011) Self-organising software: from natural to artificial adaptation. Natural computing. Springer
29. Sirin E, Parsia B, Grau BC, Kalyanpur A, Katz Y (2007) Pellet: a practical OWL-DL reasoner. Web Semant 5:51–53
30. Stovall D, Julien C (2007) Resource discovery with evolving tuples. In: International workshop on engineering of software services for pervasive environments: in conjunction with the

6th ESEC/FSE joint meeting, ESSPE '07. ACM, New York, NY, USA, pp 1–10. doi:10.1145/1294904.1294905
31. Sudeikat J, Renz W (2008) Engineering environment-mediated multi-agent systems. Springer-Verlag
32. Tolksdorf R, Menezes R (2004) Using swarm intelligence in linda systems. In: Omicini A, Petta P, Pitt J (eds) Engineering societies in the agents world IV. Lecture notes in computer science, vol 3071. Springer, Berlin/Heidelberg, pp 519–519. doi:10.1007/978-3-540-25946-6_3
33. Tolksdorf R, Nixon LJB, Simperl EPB (2008) Towards a tuplespace-based middleware for the semantic web. WIAS 6(3):235–251
34. Viroli M, Casadei M (2009) Biochemical tuple spaces for self-organising coordination. In: Coordination languages and models. LNCS, vol 5521. Springer, pp 143–162
35. Viroli M, Casadei M, Omicini A (2009) A framework for modelling and implementing self-organising coordination. In: 24th annual ACM symposium on applied computing (SAC 2009), vol III. ACM, Honolulu, Hawai'i, USA, pp 1353–1360
36. Viroli M, Casadei M, Montagna S, Zambonelli F (2011) Spatial coordination of pervasive services through chemical-inspired tuple spaces. ACM Trans Auton Adap 6(2):14:1–14:24. doi:10.1145/1968513.1968517
37. Viroli M, Nardini E, Castelli G, Mamei M, Zambonelli F (2011) A coordination approach to adaptive pervasive service ecosystems. In: IEEE international conferences on self-adaptive and self-organizing systems – workskop AWARE
38. Viroli M, Pianini D, Beal J (2012) Linda in space-time: an adaptive coordination model for mobile ad-hoc environments. In: Sirjani M (ed) Coordination languages and models. LNCS, vol 7274. Proceedings of the 14th conference of coordination models and languages (Coordination 2012), Stockholm, Sweden, 14–15 June. Springer, pp 212–229
39. Viroli M, Pianini D, Montagna S, Stevenson G (2012) Pervasive ecosystems: a coordination model based on semantic chemistry. In: Ossowski S, Lecca P, Hung CC, Hong J (eds) In: 27th annual ACM symposium on applied computing (SAC 2012). ACM, Riva del Garda, TN, Italy, pp 295–302
40. Viroli M, Zambonelli F, Stevenson G, Dobson S (2012) From soa to pervasive service ecosystems: an approach based on semantic web technologies. In: Cubo J, Ortiz G (eds) Adaptive web services for modular and reusable software development: tactics and solution, chap 8. IGI Global, pp 207–237. doi:10.4018/978-1-4666-2089-6.ch008
41. Zambonelli F, Viroli M (2011) A survey on nature-inspired metaphors for pervasive service ecosystems. Int J Pervas Comput Commun 7(3):186–204
42. Zhang W, Hansen KM (2008) Semantic web based self-management for a pervasive service middleware. In: Proceedings of the 2008 second IEEE international conference on self-adaptive and self-organizing systems. IEEE Computer Society, Washington, DC, USA, pp 245–254. doi:10.1109/SASO.2008.14. http://dl.acm.org/citation.cfm?id=1475691.1475960