On the Space-time Situation of Pervasive Service Ecosystems

Mirko Viroli
Alma Mater Studiorum – Università di Bologna, Italy
Email: mirko.viroli@unibo.it

Graeme Stevenson
University of St Andrews, UK
Email: graeme.stevenson@st-andrews.ac.uk

Abstract—**We focus on a coordination model for pervasive computing applications, tightly coupled to Semantic Web technologies to support openness and semantic reasoning. Our approach is based on the ideas of reifying the existence of services, data, and events in terms of RDF-oriented annotations maintained by local agents, and enacting global system behaviour by manipulation rules for such annotations, which resemble chemical reactions and can be seen as sequences of SPARQL queries and SPARUL updates over RDF stores. We show a minimal set of ingredients to equip this framework with space-time computing mechanisms, including reification of space-time information in terms of annotations, and a relocation service for annotations. Some examples of spatial computations in pervasive computing are given to illustrate the approach.**

I. Introduction

A *pervasive service ecosystem* is a pervasive computing system in which the various individuals that populate it – such as humans, their smartphones, software services, pervasive displays, sensors and devices spread in the environment, sources of knowledge and data – interoperate opportunistically to achieve their private goals, but are also globally governed by some infrastructure rules analogous to the “laws of nature” in natural ecosystems [22]. Following the general approach proposed in [20], [2], which we here take as a reference, we assume that the presence and activities of such individuals are continuously reflected (in the infrastructure node in which they reside) as semantic annotations, called Live Semantic Annotations (LSA). In concert, these form a global network of annotations representing the virtual counterpart of the physical ecosystem. Overall system behaviour is driven by a set of laws, called *eco-laws*, which act locally to each node, combining and manipulating annotations in a semantic way—enacting all the fine- and coarse-grained processes of LSA interaction, composition, disposal, and so on.

As a concrete implementation of this model, we focus on one that is fully-grounded 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) [21]. We use RDF as language for structuring LSAs, and the 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 [17] can be used to support scheduling and execution of eco-laws locally.

As a key contribution of this paper, which builds on top of [21], [20], we isolate a minimal set of additional middleware services that add the ability to define spatial computing activities (evolutions of distributed structures of LSAs), namely: *(i)* automatically reifying location information in each LSA, *(ii)* reifying a node's spatiotemporal state into new LSAs injected therein, and *(iii)* asynchronously relocating LSAs that have a mismatching location property. A main advantage of the proposed technique is that spatial computations can be structured in terms of chemical-resembling reactions applying semantically to LSAs, handling temporal and spatial aspects in a fully declarative way, treating spacial and temporal aspects no differently from other service properties. We believe this idea can bring new insights as to how spatial features can be added to existing middlewares, particular those which are tuple-based. As a further contribution, we discuss several examples of spatial computations useful in the context of pervasive computing scenarios.

The remainder of this paper is organised as follows: Section II sketches our pervasive ecosystem framework, Section III details the model and its RDF/SPARQL/SPARUL serialisation, Section IV introduces our support for spatial computing aspects, Section V presents example applications, and Section VI provides concluding remarks.

II. Abstract architecture

Pervasive ecosystems [22] are characterised by two main features, which influence their underlying abstract architecture and model. On one hand, they should be *situated*, namely, the activity of any software agent and the data it produces are tightly bound to the agent's physical location: this is because any behaviour should be intrinsically aware of and affect the surrounding context. Situatedness is achieved by infrastructures reifying data, knowledge, and events in the precise point (or region) of space where they pertain, and by promoting interactions based on proximity. Accordingly, a cornerstone of pervasive ecosystems is that a *uniform representation* is required for the various software agents living within them (whether they run on smartphones, sensors, actuators, displays, or any other computational device). We term such a representation a “Live Semantic Annotation” (LSA) for it should continuously represent the state of its associated component (live), and it should be implicitly or explicitly connected to the domain in which such information is produced, interpreted and manipulated (semantic). The LSAs of each agent are reified in a distributed space (called an “LSA-space”) acting as the

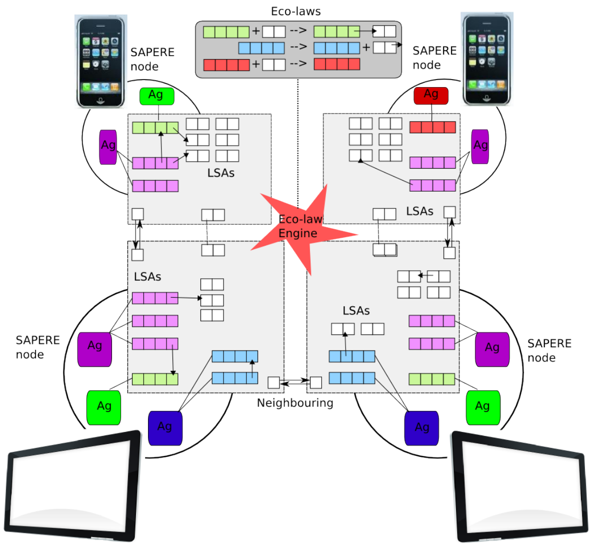

Fig. 1. An architectural view of a pervasive ecosystem.

fabric of the ecosystem, located in the computational device hosting the agent.

On the other hand, pervasive ecosystems should be *adaptive*, exhibiting properties of autonomous adaptation and management to survive contingencies without human intervention and/or global supervision. This is achieved following the natural inspiration [22], by designing system rules that – by acting locally – make global properties emerge dynamically. So, while agents enact their *individual* behaviour by observing their context and updating their LSAs, *global* behaviour (i.e., global system coordination) is enacted by *self-organising* manipulation rules of the LSA-space, which we call *eco-laws*. They can execute delete/update/create actions applied to a small set of LSAs *within the same locality*. Following [3], [18], such eco-laws are structured as chemical-resembling reactions over LSAs.

Figure 1 shows an architectural view, based on the above abstractions, of a portion of an ecosystem featuring: two public displays and two smartphones (carried by people in front of displays), forming a network of 4 computational nodes, each with a local LSA-space containing some running agents (e.g., profile agents and sensor agents in smartphones); LSAs through which agents manifest (in colour); additional LSAs representing data, knowledge, and contextual information like the existence of neighbouring nodes (in white); references from one LSA to another (also called *bonds*); and a set of eco-laws executed by an underlying engine working over the global LSA-space. More generally, one should think of a very large 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 that evolve over time. The eco-law engine, accordingly, has to be seen as a distributed one uniformly working on all LSA-space—though we will show that a feasible approach amounts at developing local eco-law engines in each node.

III. A CONCRETE MODEL OF LSAS AND ECO-LAWS

A. *Live Semantic Annotations*

LSAs have a unique, system-wide identifier (LSA-id), and a content (description) including all the information the agent wants to manifest to the ecosystem. This is realised as an RDF-like (Resource Description Framework [10]) set of multi-valued properties, or equivalently, a set of triples that consist of a subject (an LSA-id), a predicate (the property name, a Uniform Resource Identifier – URI) and an object (the assigned value, a literal, URI, or bnode[1]). URIs are qualified by universally-accessible namespaces (using syntax `namespace:term`). In RDF, a literal can be qualified by an XML Schema datatype (XSD) as in `"10.0"^^xsd:double` to enforce type-checking, but we shall omit it for the sake of simplicity, and simply consider quoted strings. By adopting a notation resembling N3 [6], an LSA is represented e.g,. as "`id p v; id 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`. Following N3, we can avoid repeating the subject when this is unchanged with respect to previous triple, hence writing "`id p v; q w1 w2 w3;`" for the above example. A concrete example of an LSA is, hence:

```
lsa:crowdsensorlsa1123
    eco:type msm:crowd;
    msm:time "2011-05-30T11:00:00";
    msm:crowd_level "0.9";
```

which is the LSA injected by a sensor that describes the precise point in time that a value (0.9) concerning the presence of people in a given room of a museum is sensed.

B. *Eco-laws*

Eco-laws are structured as chemical-resembling rules of the kind "`P+..+P --r--> Q+..+Q`". Elements `P` and `Q` are patterns of LSAs, expressed like LSAs with the following extensions: *(i)* in place of each element of a triple one can use a variable `?V` (matching any value) or an annotated variable `?V(`*`filter`*`)` where *`filter`* is a predicate expression over `?V` (matching any value that makes *`filter`* true); and *(ii)* the object of a triple can be prepended by a symbol "+" (assumed by default), "-", or "="— respectively meaning that the triples with this object should exist, should not exist, should be the only that exists for that subject and predicate. For syntactic convenience, we also allow a pattern to consist solely of the source, meaning no further constraint on its triples is imposed. Additionally, we sometimes use as filter for a subject `?LSA` an expression of the kind "`?LSA clones ?LSA2`", meaning that `?LSA` should have the same content of `?LSA2` plus additional constraints specified by any following triples. The above definition of a pattern naturally induces the concept of an LSA matching a pattern (modulo a substitution of variable to terms).

[1]A bnode, or blank node, is an locally scoped identifier. Bnodes are used to represent structured property values within LSAs, although we do not elaborate this concept within this paper.

The semantics of an eco-law is then as follows. It consumes a set of *reactant LSAs* based on left-hand side patterns and produces a set of *product LSAs* based on the right-hand side patterns. In particular, right-hand side patterns are to be seen as post-conditions applied to the selected reactant LSAs. Eco-laws also obey a numeric transformation rate `r` representing a Markovian rate in a continuous-time Markov chain (CTMC) system. Such a rate can be omitted, in which case it is assumed to be infinite, that is, the eco-law is executed with "as soon as possible" semantics.

An eco-law can apply in many different locations of the ecosystem, and to different sets of (co-located) LSAs. We call *reaction* the pair consisting of a set of reactant LSAs and their corresponding product LSAs that an eco-law can trigger. Execution of a reaction amounts to atomically removing reactant LSAs from the LSA-space and inserting product LSAs back.

As an example eco-law, consider the following, which aggregates two LSAs produced by a crowd sensor, so as to keep the most recent:

```
?LSA  eco:type msm:crowd; msm:time =?T; +
?LSA2 eco:type msm:crowd; msm:time =?T2(?T2<?T);
--r-->
?LSA
```

Note that in the right-hand side we do not specify triples for `?LSA2`, which means that the LSA with id `?LSA2` will be removed, and that for `?LSA` we simply state it will be left unchanged. Another example of an eco-law, used to make a display activate and show an advertisement as soon as the presence of a person with a matching profile is sensed (e.g., the LSA of an user present in the same space), is the following:

```
?DIS eco:type msm:display; msm:status ="ready";   +
?ADV eco:type msm:ad; msm:content ?C;             +
?USR eco:type msm:usr; msm:prof ?P(?P matches ?C);
--r-->
?DIS msm:status ="showing"; msm:service ?ADV; +
?ADV +
?USR
```

Note that object `="ready"` in the left-hand side means that `"ready"` is the only object for subject `?DIS` and predicate `msm:status`, while object `="showing"` in the right-hand side means that `"showing"` should replace any previous value, while using object `"showing"` (or `+"showing"`) would mean adding value `"showing"`. We have defined a formal mapping between eco-laws and SPARQL/SPARUL, which is not reported here for brevity. As an example, the latter eco-law is written as:

```
SELECT DISTINCT * WHERE{
  ?DIS eco:type msm:display .
  ?DIS msm:status "ready" .
  FILTER NOT EXISTS { ?DIS msm:status ?o .
                      FILTER (?o!= "ready") }
  ?ADV eco:type msm:ad .
  ?ADV msm:content ?C .
  ?USR eco:type msm:user .
  ?USR msm:prof ?P; FILTER(?P rdf:type ?C) .
}
REMOVE DATA {!DIS msm:status ?o}
INSERT DATA {!DIS msm:status "showing"}
INSERT DATA {!DIS msm:service !ASV"}
```

Put in more general terms, each eco-law is mapped to one single SPARQL `SELECT` query, and a sequence of SPARUL `REMOVE` or `INSERT` statements. The first query checks whether and how the eco-law applies, yielding a set of bindings for all the variables involved—one binding of variables per each solution found, namely, per each set of reactant LSAs. Given one binding, the SPARUL statements are used to apply the eco-law. To this end, we let a placeholder `!VAR` stand for the value to which variable `?VAR` is linked to by the binding produced by SPARQL query.

C. Architectural Components

From an implementation viewpoint, the framework can be realised as a lightweight and minimal middleware that reifies LSAs in the form of semantic tuples, to be dynamically stored and updated in a system of spatially-situated tuple spaces spread over the devices of the network. The eco-laws governing the ecosystem are deployed in all network nodes, and apply locally[2]. Each node comprises a set of modules managing LSAs and Eco-laws, described in turn, all based on functionality provided by the ARQ query engine [1] and Pellet reasoner [17].

External Interface: The interface by which agents, devices, services and other nodes – namely, the *external environment* – interacts with the node providing operations to locally `inject` a new LSA, `observe` an LSA with a known ID, and `modify` and `remove` the LSAs the agent owns (injected).

Space: A space represents a passive component, similar to a tuple space, storing LSAs that are local to the node. It is responsible for the identification of LSAs within the node, hence, it manages LSA naming through unique identifiers. It is also the module in charge of implementing any possibly sophisticated indexing algorithm and data structure with the goal of quickly retrieving and filtering candidate LSAs matching an eco-law to be executed. Considering the above-mentioned technologies, the space is easily realised as an RDF-store.

Matcher: During the processing of eco-laws, the reaction manager checks whether an eco-law can apply to a candidate set of LSAs—extracted from the space based on pattern matching. More specifically, it computes all the bindings for a given

[2]At the time of writing, one such prototype is under construction in the context of SAPERE project [2].

eco-law by executing the SPARQL query. Additionally, this component also computes all filter expressions, the evaluation of which is deferred to Pellet, which – other than standard mathematical functions – allows one to code external functions computing, e.g., any ad-hoc matching between the arguments. The semantics provided by RDF Schema (RFDS) [8] and the Web Ontology Language (OWL) [9] support vocabularies that define classes of resources, semantically-rich relations, and sets of restrictions on how both may legally be combined. Application of these vocabularies to an RDF model may be verified for correctness, and inferences – such as the classification of resources – may be drawn. Indeed, standard OWL classification provides one approach to realising ontology-based semantic matching in our eco-law language. A filter `?A matches ?C` can be interpreted as `?A rdf:type ?C`. In standard OWL semantics, this considers an individual a valid substitution for `?A` if its description satisfies the set of restrictions that describe `?C`, an OWL Class description. Hence, whenever one or more ontologies are used in the pervasive ecosystem, they must be accessible to the Matcher component.

Reaction manager: The reaction manager handles events occurring within the node, based on the set of eco-laws it holds. An event describes either an external operation upon an LSA (injection, observation, removal or modification), or the activation of a specific eco-law to be executed. Events of the first kind are considered with highest priority, and are simply processed by interaction with the space. In the second case, the event is characterised by a reaction, indicating the eco-law it refers to, its binding, and the time at which it should be executed. The reaction manager exploits a simple *scheduling engine*, maintaining a list of scheduled events sorted by their occurrence time. It takes the next event, and executes the corresponding SPARUL statements through ARQ.

IV. Adding spatial computing features

The framework described so far contains only interactions of agents working in the same node, mediated by some form of local knowledge, similarly to other coordination frameworks like those in [14], [7]. In this section we develop an arguably minimal extension, enabling the possibility to enact spatial computing, by which distributed behaviour useful to pervasive computing purposes can be supported. This is based on three ingredients we describe in turn.

LSAs reifying location: We shall assume that each LSA carries one property named `eco:#loc`, holding one URI value representing the ID of the location (i.e., the node) in which the LSA currently resides. We refer to this property as a "synthetic" one, for it is not specified by the agent that injects the LSA, but it is rather created and maintained by the infrastructure. Additionally, this is a property that cannot be changed by agents, but only by eco-laws as we will detail in the following.

Space-time LSAs: A core idea of pervasive service ecosystems is that, all information deemed important for the overall system coordination should be reified within LSAs. This applies, in fact, to all agent activities, and the data, knowledge, and events they produce. Additionally, contextual information bridging ecosystem evolution with the physical word has to be properly reified into what we call synthetic LSAs—again, "synthetic" here refers to the fact that these LSAs are maintained by the infrastructure (e.g., by some agents in charge of injecting them and keeping them updated by some timing policy). Most importantly here, this concerns the space-time situation of the computation, that is, at which time we are currently executing, and how the local space is shaped.

Accordingly, we shall first assume that in each node we have the so-called *time LSA*, an LSA carrying information about the current time in the node, which is of the kind:

```
lsa:timelsa321
   eco:type eco:#timeLSA;
   eco:#time "2011-05-30T11:00:00";
   eco:#loc sid:node34164@room132;
```

Concerning space, we reify the shape of space as can be perceived from a single node, namely, what are the neighbours in the node's proximity (possibly including their IDs, their estimated distance, the kind of connectivity, the maximum communication bandwidth, the relative orientation in space, and any other information the infrastructure can discern). In particular, in any node, we assume that for each neighbour there is a synthetic *neighbour LSA* of the kind

```
lsa:neighlsa456
   eco:type eco:#neighbourLSA;
   eco:#loc sid:node34164@room132;
   eco:remotelocation sid:node34163@room132;
   eco:distance "51.3";
   eco:orientation "north-east";
```

stating that the neighbouring node is at location `node34163@room132`, which is at expected distance 51.3 (meters) in a north-easterly direction.

We reiterate that the connection between such LSAs and the neighbours they represent is entirely implicit; nodes do not directly manage their remote representations, and these synthetic LSAs are not proxies through which information from remote spaces may be obtained.

Relocation service: One of the motivations for reifying these synthetic LSAs is the ability, by means of proper eco-laws, to make their actual content impact ecosystem dynamics, by which we can fully achieve context-dependent behaviour. Concerning space, for instance, one can develop an eco-law that changes the value of the `eco:#loc` property of an LSA, replacing it with that of a neighbouring node. One such eco-law is:

```
?LSA eco:type msm:crowd; eco:status ="tomove"; +
?NEI eco:type eco:#neighbourLSA; eco:remote ?L;
--r-->
?NEI + ?LSA eco:#loc =?L; eco:status -"tomove";
```

This causes an LSA to have a location that no longer fits the current node in which it resides. A lower-level middleware service then, can be in charge of intercepting such LSAs before they are injected in the space, and relocating them to the proper neighbour, by an asynchronous request.

Note that our management of space-time aspects has the advantage of being fully declarative—e.g., an LSA with a new location can just be perceived as the LSA having been relocated. By this approach we retain the spatial locality property of eco-laws, mitigating the need to perform synchronisation across spaces during their application. This approach also orthogonally supports more advanced concepts of "topology", such as notions of social neighbouring (connecting smartphones of people who are friends on a social network as envisioned in [16]).

V. Examples

While a basic use of the above ingredients supports relocating information to mirror its physical counterpart's movements, they also permit the enactment of some patterns of spatial computing that enable useful interactions within pervasive service ecosystems; here we describe 4 such patterns, accompanied by illustrative examples.

A. Gossiping

We first show how we can make an LSA spread from a given location to all the nodes of the network, with the further ability of expiring everywhere at a given time. This is the set of eco-laws realising this behaviour:

```
[GOS] % ?GOS gets spread in any neighbour
?GOS spc:type spc:goss; +
?NEI eco:type eco:#neighbourLSA; eco:remote ?L;
--r-->
?NEI + ?GOS + ?CLO(?CLO clones ?GOS) eco:#loc =?L;

[AGG] % Of two similar LSAs, it removes one
?AGG spc:type spc:aggr; spc:content ?C; +
?AG2 spc:type spc:aggr; spc:content ?C;
--> ?AGG

[DEL] % Disposes an LSA if its deadline expired
?DEL spc:type spc:del; spc:deadline = ?T;    +
?TIM eco:type eco:#timeLSA; eco:time ?T2(?T<?T2);
--> ?TIM
```

Initially, a gossip LSA with spc:type set to values spc:goss, spc:aggr, and spc:del is injected in a node—so that all three eco-laws apply. The former eco-law makes any gossip LSA ?GOS (having property spc:type set to spc:goss) create a cloned version ?CLO relocated in a neighbour node ?L. Iterative application of this eco-law over time (at rate r) and over all nodes makes copies of ?GOS flood the network. The second eco-law takes two gossip LSAs (of kind spc:aggr) with same content (namely, relative to the same source) and drops one: this is used to avoid multiple versions residing in the same node. Finally, the latter eco-law fires when the current time in a node is greater than the deadline time spc:deadline defined in the gossip LSA, causing removal of that LSA. Of course, rate r is to be properly designed to tune the network load, using considerations similar to those discussed in [18]. Note that the three eco-laws orthogonally apply—one could leverage spreading without time-disposal, or time-disposal alone, and so on.

In the context of pervasive computing applications, this pattern can be useful to advertise an event happening in a given node to the whole network. Considering e.g., the application scenario in [12], it could be used to advertise a fire alarm in an exhibition centre, so as to immediately trigger all the activities necessary to safely steer people towards exits.

B. Gradient

A variation of the above diffusion mechanism can be used to create a gradient data structure—a key brick of several spatial computing patterns [18], [4], [19]. This is based on the idea of spreading copies of an LSA such that each of them holds the estimated distance from the source according to the shortest path. We shall also equip each LSA with a reference to the next node to traverse in order to follow the gradient towards the source, and provide a mechanism to limit the gradient to a given spatial extent (gradient horizon). This is the set of eco-laws realising this behaviour:

```
[GRA] % ?GRA gets spread with increasing spc:dist
?GRA spc:type spc:gra; spc:dist ?D;
     spc:rng ?R; eco:#loc ?LG; spc:source ?S +
?NEI eco:type eco:#neighbourLSA; eco:remote ?L;
     eco:distance ?D2(?D+?D2<=?R);
--r-->
?NEI + ?GRA +
?CLO(?CLO clones ?GRA) eco:#loc =?L; eco:prev =?LG;
     spc:dist =?D3(?D3 = ?D+?D2);

[SHR] % Of two paths, the shortest one is kept
?GRA spc:type spc:short; spc:content ?C;
     spc:dist ?D; +
?GR2 spc:type spc:short; spc:content ?C;
     spc:dist ?D2(?D2>?D);
--> ?GRA
```

Initially, a gradient LSA with spc:type set to spc:gra and spc:short, and spc:dist set to 0 is to be injected in a node; we also assume (for the sake of the following examples) that property spc:source is initially set to the LSA-id itself—by spreading, this property will hold everywhere the ID of the LSA from which the gradient originates. The former eco-law creates a cloned version ?CLO relocated in a neighbour node ?L, with an increased value of distance depending on the estimated distance of that neighbour—note this does not happen if horizon ?R is escaped. The second eco-law takes two gradient LSAs with same content and keeps the one with smaller distance.

In the context of pervasive computing applications, this pattern can be useful to advertise an event in a given node with additional information on how its source can be reached. E.g., as a fire alarm has been spread, people can be directed to any exit (acting as gradient source) by signs appearing in their smartphone or on public displays, properly reflecting the direction to take as can be inferred from property `eco:prev` of gradient LSAs.

C. Partitioning

This pattern is used to partition a network in n areas once n nodes (sources) have been selected as candidate centres of these areas, ensuring that these areas have also similar size whenever possible. This is achieved by making each source spread a different gradient, such that propagation does not overlap in nodes in which other gradients are already established with smaller distance to another source. In this way, each node will belong to the area of the nearest source node. We observe that the above eco-laws for gradients already support this behaviour, provided that the n source LSAs share the same `spc:content`, but have e.g., different values of a property `spc:area`.

An example application can be envisioned in the context of adaptive pervasive displays [18]. Assuming we have n different advertisements to show in an airport, and we want the set of displays showing one of them to form a contiguous area (to avoid people perceiving different advertisements as the pass by neighbouring displays), we can use the partition pattern and let displays choose what to visualise depending on the area they belong to. Note this pattern automatically accommodates the injection and removal of sources.

D. Path

Another pattern proposed in [11] is the path connecting two distinct nodes, possibly enlarged to the set of devices whose distance from that path is smaller than a given horizon h. This is achieved by first making the two nodes (called source and target) create their own gradient (with sufficient horizon to reach each other). As soon as the source perceives the target, it should gossip a new LSA indicating their relative distance d. Then, each node should compare d with the sum of its distance to source and target: if the difference is smaller than h, then a new LSA is created to tag this node as being part of the path. The eco-laws realising this behaviour are as follows:

```
[PATH] % SRC senses TRG gradient, gossiping distance
?GRA spc:type spc:gra spc:pathtrg;
     spc:dist ?D; spc:source ?TRG; +
?SRC eco:type spc:gra spc:pathsrc; spc:dist 0;
--r-->
?GRA + ?SRC +
?GSP(?GSP clones ?GRA)
     eco:type =spc:goss =spc:aggr;
     spc:pathtrg =?TRG; spc:pathsrc =?SRC:

[SUM] % Reifying ?PTH if the node is inside the path
?TRG spc:type spc:gra; spc:dist ?DT; +
?SRC spc:type spc:gra; spc:dist ?DS; spc:rng = ?R +
?GSP spc:pathtarget =?TRG; spc:pathsrc =?SRC;
     spc:dist ?DP(?DP>?DT+?DS-?R);
--->
?TRG + ?SRC + ?GSP +
?PTH(?PTH clones ?GSP) spc:type =spc:path =spc:shr;
```

The former eco-law makes a source detect the target at distance `?D` and correspondingly gossip (without timeout) that value of distance. The second eco-law makes any node inside the path creating an LSA of kind `spc:path`. Note the latter LSA is also of type `spc:shr`, so that only the copy with shortest distance is kept.

This pattern can be useful to mark the transiting area of people steered from a place to another in an articulated environment, as in the case of people moving from one gate to another in an airport. Public displays can be programmed to show signs towards the target gate only if they stay inside the path dynamically computed as described above. In this way, we avoid affecting all the displays of the airport, but may still handle the case of people departing slightly from the optimal path.

VI. Conclusions and Future Work

In this paper we described how spatial computing concepts can be injected into a framework of pervasive service ecosystems, in terms of a minimal set of features concerning reification of space-time information (namely, bridging the gap between the computational world and the physical world), and a relocation service designed to remedy mismatches between the locality declared by some information and its actual position. Several examples of applications to spatial computing structures are provided to explain the details of these mechanisms and show the usefulness in pervasive computing. We believe that the work presented in this paper has a validity beyond the pervasive ecosystem framework, for it can be smoothly applied to any distributed system based on a notion of shared data-space (contrasting approaches based on message-passing such as e.g. [11]).

The proposed chemical model can be extended in several ways, all of which will be subject of our future investigations. First of all, we currently retain a quite rigid structure in which the number of reactants and products is statically defined: further studies are needed to understand to which extent the corresponding language can mimic transformations working over sets of LSAs whose size is not known a priori. Then, we currently rely on CTMC semantics to trigger eco-laws, without considering further priority constructs which could be interesting. Also, we assume a flat set of LSAs without any hierarchical structuring of the topological space, which would be of some interest for pervasive computing applications. Finally, we note that a distributed setting requires security and privacy mechanisms that complement the openness of the system. We intend to develop such mechanisms as part of the ecosystem fabric: for example, supporting spatially restricted information flow and using inference based on property and concept hierarchies to appropriately abstract information viewable by agents. Alternatively, general frameworks tackling security in coordination models like [13] can be evaluated.

A further roadmap for future works aimed at strengthening the relationships between spatial computing, space-based coordination models, and their applicability to pervasive computing applications, includes the following activities: *(i)* identifying a concept of expressiveness of spatial computations and a minimal set of mechanisms to achieve it, along the lines of [5]; *(ii)* studying techniques for predicting and controlling the global behaviour that emerges out of the local coordination rules; *(iii)* deepening the advantages of using semantic matching in the context of spatial computing patterns, as e.g., exploited in [15]; *(iv)* thoroughly analysing the applicability of spatial computing to emerging ICT scenarios like smart cities, intelligent traffic control, and augmented social reality.

Acknowledgments

This work has been supported by the EU FP7 project "SAPERE - Self-aware Pervasive Service Ecosystems" under contract No. 256873.

References

[1] ARQ - a SPARQL processor for Jena. http://jena.sourceforge.net/ARQ/, 2011.

[2] Self-aware pervasive service ecosystems. http://www.sapere-project.eu, 2012.

[3] J.-P. Banâtre and T. Priol. Chemical programming of future service-oriented architectures. *JSW*, 4(7):738–746, 2009.

[4] J. Beal. Flexible self-healing gradients. In *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC)*, pages 1197–1201. ACM, 2009.

[5] J. Beal. A basis set of operators for space-time computations. In *Self-Adaptive and Self-Organizing Systems Workshop (SASOW 2010)*, pages 91 –97, sept. 2010.

[6] T. Berners-Lee and D. Connolly. Notation3 (N3): A readable rdf syntax. W3C team submission, W3C, 2011. http://www.w3.org/TeamSubmission/n3/.

[7] G. Cabri, L. Leonardi, and F. Zambonelli. MARS: A programmable coordination architecture for mobile agents. *IEEE Internet Computing*, 4(4):26–35, 2000.

[8] R. V. Guha and D. Brickley. RDF vocabulary description language 1.0: RDF schema. W3C recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[9] M. Krötzsch, P. F. Patel-Schneider, S. Rudolph, P. Hitzler, and B. Parsia. OWL 2 web ontology language primer. Technical report, W3C, Oct. 2009. http://www.w3.org/TR/2009/REC-owl2-primer-20091027/.

[10] E. Miller and F. Manola. RDF primer. W3C recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

[11] MIT Proto. software available at `http://proto.bbn.com/`, Retrieved Nov. 1st, 2010.

[12] S. Montagna, M. Viroli, M. Risoldi, D. Pianini, and G. Di Marzo Serugendo. Self-organising pervasive ecosystems: A crowd evacuation example. In *Workshop on Software Engineering for Resilient Systems*, volume 6968 of *LNCS*, pages 115–129. Springer, 2011.

[13] A. Omicini, A. Ricci, and M. Viroli. An algebraic approach for modelling organisation, roles and contexts in MAS. *Applicable Algebra in Engineering, Communication and Computing*, 16(2-3):151–178, Aug. 2005.

[14] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, Sept. 1999.

[15] D. Pianini, S. Virruso, R. Menezes, A. Omicini, and M. Viroli. Self organization in coordination systems using a WordNet-based ontology. In *4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2010)*, pages 114–123. IEEE CS, 27 Sept.–1 Oct. 2010.

[16] A. Rosi, M. Mamei, F. Zambonelli, S. Dobson, G. Stevenson, and J. Ye. Social sensors and pervasive services: Approaches and perspectives. In *PerCom Workshops*, pages 525–530. IEEE, 2011.

[17] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semant.*, 5:51–53, June 2007.

[18] M. Viroli, M. Casadei, S. Montagna, and F. Zambonelli. Spatial coordination of pervasive services through chemical-inspired tuple spaces. *ACM Transactions on Autonomous and Adaptive Systems*, 6(2):14:1 – 14:24, June 2011.

[19] M. Viroli, D. Pianini, and J. Beal. Linda in space-time: an adaptive coordination model for mobile ad-hoc environments. In M. Sirjani, editor, *Proceedings of the 14th Conference of Coordination Models and Languages (Coordination 2012),Stockholm (Sweden), 14-15 June*, Lecture Notes in Computer Science. Springer, 2012.

[20] M. Viroli, D. Pianini, S. Montagna, and G. Stevenson. Pervasive ecosystems: a coordination model based on semantic chemistry. In S. Ossowski, P. Lecca, C.-C. Hung, and J. Hong, editors, *27th Annual ACM Symposium on Applied Computing (SAC 2012)*, Riva del Garda, TN, Italy, 26-30 March 2012. ACM.

[21] M. Viroli, F. Zambonelli, G. Stevenson, and S. Dobson. *From SOA to Pervasive Service Ecosystems: an approach based on Semantic Web technologies*. IGI Global, 2012. Available for reviewers to download at: http://apice.unibo.it/xwiki/bin/download/Publications/SemanticSapereIGI2012/chapter.pdf.

[22] F. Zambonelli and M. Viroli. A survey on nature-inspired metaphors for pervasive service ecosystems. *International Journal of Pervasive Computing and Communications*, 7(3):186–204, 2011.