From SOA to Pervasive Service Ecosystems: an approach based on Semantic Web technologies

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

Franco Zambonelli
Università di Modena e Reggio Emilia, Italy
franco.zambonelli@unimore.it

Graeme Stevenson, Simon Dobson
University of St Andrews, UK
{graeme.stevenson, simon.dobson}@ st-andrews.ac.uk

ABSTRACT

Emerging pervasive computing scenarios require open service frameworks promoting situated adaptive behaviors and supporting diversity in services and long-term ability to evolve. We argue that this calls for a nature-inspired approach in which pervasive services are modeled and deployed as autonomous individuals in an ecosystem of other services, data sources, and pervasive devices. We discuss how standard service-oriented architectures have to evolve to tackle the above issues, present a general architecture based on a shared spatial substrate mediating interactions of all the individual services of the pervasive computing system, and finally show that this architecture can be implemented relying primarily on standard W3C Semantic Web technologies, like RDF and SPARQL. A use case of adaptive pervasive displays for crowd steering applications is exploited as reference example.

1. INTRODUCTION

The ICT landscape, notably changed by the advent of ubiquitous wireless connectivity, is further reshaping due to the increasing deployment of pervasive computing technologies. Via RFID tags and similar technologies, objects will carry a wide range of digital, self-describing information. Wireless sensor networks and camera networks will spread across our cities and buildings to monitor physical phenomena. Smartphones and other personal devices will increasingly sense and store notable amounts of data related to our personal, social and professional activities, beyond feeding (and being fed by) the Web with spatial and social real-time information (Campbell et al., 2008).

This evolution is contributing to building integrated and dense infrastructures for the pervasive provisioning of general-purpose digital services. If all their components are able to opportunistically connect with each other, such infrastructures can be used to enrich existing services with the capability of autonomously adapting their behavior to the physical and social context in which they are invoked, and will also support innovative services for enhanced interactions with the surrounding physical and social worlds (Coleman, 2009).

Users will play an active role by contributing data and services and by making available their own sensing and actuating devices. This will make pervasive computing infrastructures as participatory and as capable of value co-creation as the Web (Spohrer et al., 2007), eventually acting as globally shared substrates to externalize, enhance, and make more valuable our physical and social intelligence.

We already face the commercial release of a variety of early pervasive services trying to exploit the possibilities opened by these new scenarios: GPS navigation systems providing real-time traffic information and updating routes accordingly, cooperative smartphones that inform us about the current positions of our friends, and augmented reality services that enrich what we see around with dynamically retrieved digital information (Ferscha and Vogl, 2010). However, the road towards the effective and systematic exploitation of these emerging scenarios calls for a radical rethinking of current service models and frameworks.

Elaborating on this problem, this chapter is organized as follows:

- In Section 2 we provide a background: starting from a case study of adaptive pervasive displays, we discuss the basic requirements of emergent pervasive computing applications, namely situatedness, adaptation, and support for diversity and long-term evolution;
- In Sections 3-5 we present the main focus of the chapter: in Section 3.1 we propose how current SOA solutions are to be evolved to tackle those requirements, namely, by a deep rethinking inspired by nature and its mechanisms, promoting the idea of a "pervasive and shared spatial continuum" over which local individual interactions occur; and in Section 3.2 we present an innovative architecture for pervasive service ecosystems rooted in the concepts of "Live Semantic Annotations" (LSAs, representing interfaces of environment services) and "eco-laws" (global co-ordination rules enabling and regulating interactions). In Section 4 an implementation of the framework based on standard W3C technologies for the Semantic Web is discussed, namely, relying on RDF for supporting LSAs, and SPARQL for eco-laws; Section 5 presents a concrete example in the form of a crowd steering example.
- In Section 6 we discuss related works.
- Finally, in Section 7, we conclude by describing the future directions of the presented research.

2. BACKGROUND

As background for this chapter, we present the pervasive computing scenarios we intend to target, so as to emphasize the requirements they pose. We do this by means of a simple case study – representative of a larger class of emerging pervasive scenarios – which will be used to ground our arguments, sketch the requirements of future pervasive services, and ultimately discuss the proposed service framework.

It is a matter of fact that we are increasingly surrounded by digital displays: from those of wearable devices to wide wall-mounted displays pervading urban and working environments. Currently, the former interact with the environment and are affected by its contingencies in limited manners, while the latter are simply conceived as static information servers to show information in a manually-configured manner – e.g., cycling some pre-defined commercials or general interest news – independently of both the context in which they operate and of nearby users. However, such overall display infrastructures can be made more effective and advantageous for both users and information/service providers by becoming general, open, and adaptable information service infrastructures.

First, information should be displayed based on the current state of the surrounding physical and social environment. For instance, by exploiting information from sensors and from profiles of nearby users, a display sited within a museum could select an exhibition to advertise based on overlapping user interests. Also, actions could be coordinated among neighboring displays to enact different policies, e.g., to avoid irritating users with the same ads as they pass by, to combine adjacent displays for the purpose of presenting complex, multifaceted information, or to coordinate displays to steer people towards an exhibition. These examples express a general requirement for pervasive services:

- *Situatedness* — Pervasive services deal with spatially- and socially-situated activities of users, and should thus be able to interact with the surrounding physical and social world and adapt their behavior accordingly. The infrastructure itself, deeply embedded in the physical space, should effectively deal with spatial concepts and data.

Second, and complementary to the above, the display infrastructure and the services within should automatically adapt to their own modifications and contingencies in an automatic way without malfunctioning, and possibly take advantage of such modifications. Namely, when new devices are deployed, new information is injected, or new people arrive, a spontaneous re-distribution and re-shaping of the overall displayed information should take place. For instance: the route towards an exhibition could be automatically computed by self-organization so as to dynamically avoid overcrowded rooms or corridors, or alternative exhibitions could be selected if one has reached (or will shortly reach) capacity. In terms of a general requirement for decentralized and dynamic scenarios:

- *Adaptivity* — Pervasive services and infrastructures should inherently exhibit properties of autonomous adaptation and management, to survive contingencies without human intervention and at limited costs.

Third, the display infrastructure should enable users – other than display owners – to upload information and services to enrich the offerings available, or adapt the infrastructure to their own needs. For instance, users may continuously upload personal content (e.g., pictures and annotations related to the local environment) from their own devices to the infrastructure, both for better visualization and for increasing the overall local information offer. Similarly, a group of friends can exploit a public display by uploading software letting the display host a shared real-time map, visualizing what's happening nearby (which would also require opportunistic access to the existing environmental sensors and to any available user-provided sensors, to make the map alive and rich in real-time information). In general, one should enable users to act as "prosumers" – i.e., as both consumers and producers – of devices, data, and services. This will not only make environments meet the specific needs of any user (and capture the long tail of the market), but will also induce a process of value co-creation increasing the overall intrinsic value of the system and of its services (Vargo et al., 2008). If the mentioned real-time map accesses some sensors in unconventional ways to better detect situations around, this adds value both to such sensors and to all existing and future services requiring situation recognition. In terms of a general requirement:

- *Prosumption and Diversity* — The infrastructure should tolerate open models of service production and usage without limiting the number and classes of services provided, and rather taking advantage of the injection of new services by exploiting them to improve and integrate existing services whenever possible, and add further value to them.

Finally, besides short-term adaptation, in the longer-term any pervasive infrastructure will experience dramatic changes related to the technology being adopted, as well as in the kinds of services being deployed and in their patterns of usage. For instance, a display infrastructure will at some time integrate more sophisticated sensing means than we have today and will possibly integrate enriched actuators via which to attract user attention and interact with them. This can be the case of personal projection systems to make any physical object become a display, or of eyeglass displays for immersive perception and action. While this can open up the way for brand new classes and generations of services to be conceived and deployed, it also requires that such evolution can be gradually accommodated without harming the existing infrastructure and services. As a general requirement:

- *Eternity* — The infrastructure should tolerate long-term evolutions of structure, components, and usage patterns, to accommodate the changing needs of users and technological evolution without forcing significant and expensive re-engineering efforts to incorporate innovations and changes.

3. PERVASIVE SERVICE ECOSYSTEMS

3.1 From SOA to Nature-Inspired Pervasive Service Ecosystems

Can the above requirements be met by architecting pervasive service environments around standard service-oriented architectures (SOA)? To some extents, yes, but the final result would be a sort of scramble of current SOA methodology. Indeed, we believe the above requirements call for re-thinking some assumptions of SOA architecture, and evolving them with ideas and mechanisms proposed in the field of self-organizing computer systems, namely, relying on a nature-inspired approach as developed in the following.

3.1.1 Centralized SOA Solution

In general, SOA consider the inter-related activities of service components to be managed by various infrastructural (middleware) services such as: discovery services to help components find each other; context services to help components situate their activities; orchestration services to coordinate interactions according to specific application logics; and shared data-space services to support data-mediated interactions (Wells et al., 2008). To architect a pervasive display environment in such terms (Figure 1- top), one has to set up a middleware server in which to host all the necessary infrastructural services to support the various components of the scenario, i.e., displays, information and advertising services, user-provided services, sensing devices and personal devices.

Such components become aware of each other via the discovery service. However, in dynamic scenarios (users and devices coming and going), components are forced to continuously access (or be notified by) the discovery service to preserve up-to-date information—a computational and communication wasting activity. Also, since discovery and interactions among components have to rely on spatial information (i.e., a display is interested only in the users and sensors in its proximity), this requires either sophisticated context-services to extract the necessary spatial information about components, or to embed spatial descriptions for each component into its discovery entry, again inducing frequent and costly updates to keep up with mobility.

To adapt to situations and contingencies, components should be able to recognize relevant changes in their current environment and plan corrective actions in response to them, which again requires notable communication and computational costs for all the components involved. Alternatively, or complementary, one could think of embedding adaptation logics into a specific server inside the middleware (e.g., in the form of autonomic control managers (Kephart and Chess, 2003)). However, such logics would have to be very complex and heavyweight to ensure capability of adapting to any foreseeable situation, and especially hard for long-term adaptation.

3.1.2 Decentralized SOA Solution

To reduce the identified complexities and costs and better match the characteristics of the scenario, one could think of a more distributed solution, with a variety of middleware servers deployed in the infrastructure to serve, on a strictly local basis, only a limited portion of the overall infrastructure. For instance (Figure 1 - bottom), one could install a single middleware server for each of the available public displays. It would manage the local display and all local service components, thus simplifying local discovery and naturally enforcing spatial interactions. Adaptation to situations is made easier, thanks to the possibility of recognizing in a more confined way (and at reduced costs) local contingencies and events, and of acting locally upon them.

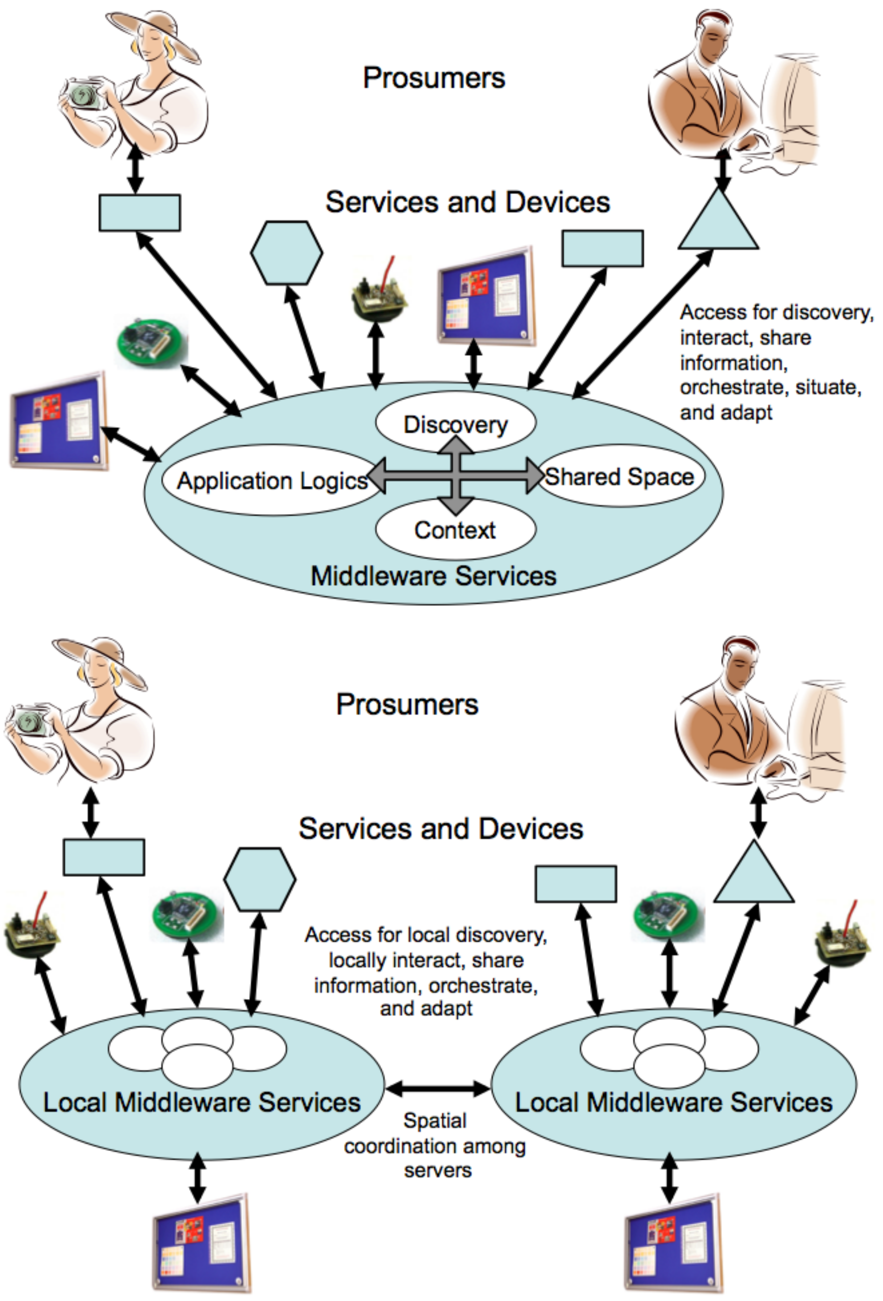
Prosumers
Services and Devices
Access for discovery, interact, share information, orchestrate, situate, and adapt
Discovery
Application Logics
Shared Space
Context
Middleware Services
Prosumers
Services and Devices
Access for local discovery, locally interact, share information, orchestrate, and adapt
Local Middleware Services
Spatial coordination among servers
Local Middleware Services

Figure 1: Architecting pervasive service environments. Top: a solution with a centralized middleware server. Bottom: a solution relying on a distributed set of local middleware services.

With the adoption of a distributed solution enforcing locality, the distinction between the logics and duties of the different infrastructural services fades: discovering local services and devices implies discovering something about the local context; the dynamics of the local scenarios, as reflected in the local discovery tables, makes it possible to have components indirectly influence each other (being that their actions are possibly dependent on such tables), as in a sort of shared data-space model. This also induces specific orchestration patterns for components based on the local logics upon which the middleware relies to distribute information and events among components and to put components in touch with each other.

A problem of this distributed architecture is that it requires solutions both to tune it to the spatial characteristics of the scenario and to adaptively handle contingencies. The logics of allocation of middleware servers (i.e., one server per display) derives naturally only in a static scenario, but the arrival and departure of displays requires the middleware servers to react by re-shaping the spatial regions and the service components of pertinence of each of them, notifying components correspondingly. Dynamism is compulsory, moreover, if one decides to install those middleware services on smartphones as well (e.g., deployed as apps), in order to seemingly handle private displays on them. To tackle this general problem, the actual distribution of middleware servers should become transparent to service components—they should not worry about where servers are, but will simply act in their local space confident that there are servers to access. Moreover, the network of servers should be able to spontaneously re-organize its shape in autonomy, without directly affecting service components but simply adaptively inducing in them a re-organization of their interaction patterns.

3.1.3 Nature-inspired eco-systems

Pushed towards a very dense and mobile network of nodes, pervasive devices and people's smartphones, the architecture will end up being perceivable as a dense distributed environment above which a very dynamic set of spatially-situated components discover, interact, and coordinate with each other. This is done in terms of much simplified logics, embedded into the unique (though highly distributed) infrastructural service, subsuming the roles of discovery, context, data-space, and orchestration services, and taking the form of a limited set of local rules embedded in the spatial substrate itself—deployed in each computational node. That is, we would end up with something that notably resembles the architecture of natural ecosystems: a set of spatially situated entities interacting according to a well-defined set of natural laws enforced by the spatial environment in which they situate, and adaptively self-organizing their interaction dynamics according to its the shape and structure.

Going further than architectural similarity, the natural metaphor can be adopted as the ground upon which to rely to inherently accommodate the requirements of pervasive service scenarios. Situatedness and spatiality are there by construction. Adaptation can be achieved because of the basic rules of the game: the dynamics of the ecosystem, as determined by the enactment of laws and by the shape of the environment, can spontaneously induce forms of adaptive self-organization beside the characteristics of the individual components. Accommodating new and diverse component species, even towards a long-term evolution, is obtained by making components party to the game in respect of its rules, and by letting the ecosystem dynamics evolve and re-shape in response to the appearance of such new species. This way, we can take advantage of the new interaction possibilities of such new services and of the additional value they bring, without requiring individual components or the infrastructure itself (i.e., its laws and structure) to be re-engineered (Jazayeri, 2005).

Indeed, nature-inspired solutions have already been extensively exploited in distributed computing (Babaoglu et al., 2006) for the implementation of specific adaptive algorithmic solutions or of specific adaptive services. Also, many initiatives – like those named upon digital/business service ecosystems

(Ulieru and Grobbelaar, 2007) – recognize that the complexity of modern service systems is comparable to that of natural ones and requires innovative solutions also to effectively support diversity and value co-creation. Yet, the idea that natural metaphors can become the foundation on which to fully re-think the architecture of service systems is far from being metabolized.

3.2 A Reference Conceptual Architecture

The above discussion leads to the identification of a reference conceptual architecture for nature-inspired pervasive service ecosystems (see Figure 2).

The lowest level is the concrete physical and digital ground on which the ecosystem will be deployed, i.e., a dense infrastructure (ideally a continuum) of networked computing devices and information sources. At the top level, prosumers access the open service framework for using/consuming data or services, as well as for producing and deploying in the framework new services and new data components or for making new devices available. In our case study, they include the users passing by, the display owners, information providers, and the advertising companies interested in buying commercial slots. At both levels openness and its dynamics arise: new devices can join/leave the system at any time, and new users can interact with the framework and can deploy new services and data items on it. In our case study, we consider integration at any time of new displays and new sensors, and the presence of a continuous flow of new visualization services (e.g., commercial advertisers) and users, possibly having their own devices integrated in the overall infrastructure.

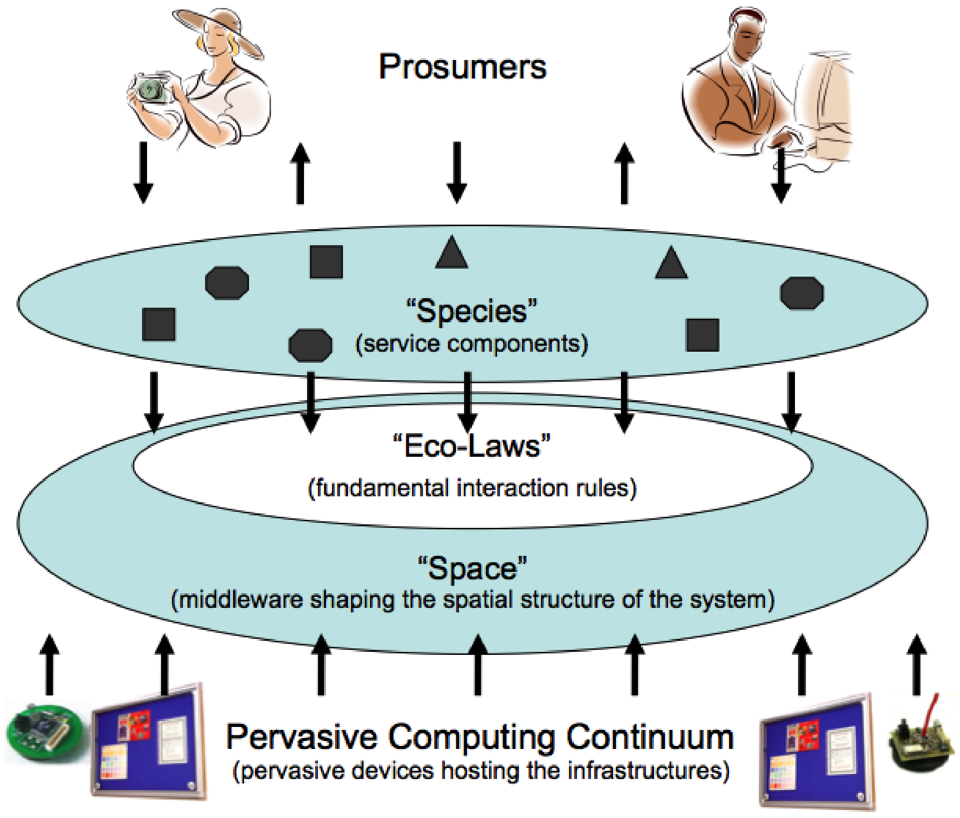
Prosumers
"Species"
(service components)
"Eco-Laws"
(fundamental interaction rules)
"Space"
(middleware shaping the spatial structure of the system)
Pervasive Computing Continuum
(pervasive devices hosting the infrastructures)

Figure 2: A Conceptual Architecture for Pervasive Service Ecosystems

In between these two levels, lie the abstract computational components of the pervasive ecosystem architecture.

- **Species** — This level includes a variety of components, belonging to different “species” yet modeled and computationally rendered in a uniform way, representing the *individuals* (i.e., software agents) populating the ecosystem: physical and virtual devices of the pervasive infrastructure, digital and network resources of any kind, providers of persistent/temporary knowledge/data and contextual information, software services, or personal user agents. In our case study, we will have different software species to represent displays and their displaying service, the various kinds of sensors distributed around the environment and the data they express, software agents to act on behalf of users, display owners, and advertisers.

  In general terms, an ecosystem is expected to be populated with a set of software agents physically deployed in the environment, situated in some portion of the ecosystem space, and dynamically joining/leaving it. The very occurrence of such agents is reified in our architecture in terms of so-called Live Semantic Annotations (LSAs), which play the role of ecosystem individuals. An LSA is a structured, semantically founded, and continuously updated annotation reflecting some relevant information for the coordination of an ecosystem, concerning events/state/interface/goal of software agents—thus conceptually extending/embedding know concepts of service interface and behavior (as e.g., in WSDL and BPEL technologies). So, any entity, state, situation or event concerning an agent is manifested by it in the form of one or more LSAs.

- **Space** — This level gives shape to the spatial fabric supporting LSAs, namely, the spatial activities and interactions of agents. Given the spatial nature of pervasive services (as it is the case of information and advertising services in our case study), this level situates individuals in a specific portion of the space, so that their activities and interactions are directly dependent on their positions and on the shape of the surrounding space.

  Practically, the spatial structure of the ecosystem will be reified by a middleware substrate, deployed on top of the physical deployment context, supporting the execution and life cycle of individuals and their spatial interactions in each computational node of the network. From the viewpoint of such individuals, the middleware will have to provide them (via some API) with the possibility of advertising themselves by LSAs, accessing information about their local spatial context (including other nearby individuals) and detecting local events via a mechanism of LSA *bonding*—an LSA can bond to others, and an agent can access information expressed within those LSAs it is connected to by bonds. From the viewpoint of the underlying infrastructure, the middleware should provide for transparently absorbing dynamic changes and the arrival/departure of supporting devices, without affecting the perception of the spatial environment by individuals.

  Technologically, this can be realized by a network of active data-oriented and event-oriented localized services, spread across the nodes of the pervasive substrate, and accessible on a location-dependent basis by individuals and devices. In the case study, for instance, one could think of assigning one such middleware service to each display and each smartphone, and have the various displays dynamically self-configure their spatial domain of competence accordingly to geographic and “line of sight” factors.

- **Eco-Laws** — The way in which individuals (whether service components, devices, or generic resources) live and interact is determined by the set of fundamental “eco-laws” regulating the ecosystem model. Enactment of eco-laws on individuals will typically affect and be affected by the local space and by the local individuals around. In our case study, eco-laws might provide for automatically and dynamically determining to display specific information on a screen as a sort of automatic reaction to specific environmental conditions, or support having two displays spontaneously aggregate and synchronize with each other in showing specific advertisements.

> An eco-law works as a sort of chemical law: it takes a set of reactant LSAs existing in the LSA-space, and substitutes it with a set of product LSAs—which could be placed either locally or in the neighborhood. Alternatively, an eco-law is a pattern of atomic transformations of groups of LSAs residing in the same locality, with an implicit/explicit scheduling policy and scope depending on the specific operational model. Such transformations can lead to: creation of new LSAs (e.g., representing composed services), deletion of existing LSAs (e.g., disposing of services no longer used), semantic manipulation of LSAs (e.g., changing a service state depending on extraction of some contextual information), creation/deletion of bonds (e.g., to connect two LSAs to make their agents interact), relocating/diffusing LSAs (e.g., to make the existence of a service be perceived in a region of space).

The proposed architecture represents a radically new perspective on modeling service systems and their infrastructures. An un-layered universe of components, all of which underlie the same model, living and interacting in the same spatial substrate, and obeying the same eco-laws—the latter being the only concept hardwired into the system, subsumes the typically multifaceted layers of SOA.

This rethinking is very important to ensure adaptation, diversity, and long-term evolution: no component, service or device is there to stay, everything can change and evolve, self-adapting over space and time, without undermining the overall structure and assumptions of the ecosystem. That is, by conceiving the middleware in terms of a simple spatial substrate in charge of enforcing only basic interaction rules, we have moved away from the infrastructure itself needing to adapt, and fully translated this as a property of the application level and of its dynamics.

Interactions between components, then, are based solely on the existence of a bond (or bonds) between their LSAs. Once established, all interaction is indirect, namely, by observing changes made to the structure and content of each partner's LSA. This embedding of core interaction principles within the environment represents a shift away from traditional coordination models, where interactions are externally shaped and executed.

The dynamics of the ecosystem will be determined by individuals acting based on their own goals/attitudes, yet being subject to the eco-laws for their interactions with others. Typical patterns that can be driven by such laws may include forms of adaptive self-organization (e.g., spontaneous service aggregation or service orchestration, where eco-laws play an active role in facilitating individuals to spontaneously interact and orchestrate with each other, also in dependence of current conditions), adaptive evolution (changing conditions reflected in changes in the way individuals in a locality are affected by the eco-laws) and of decentralized control (to affect the ecosystem behavior by injecting new components into it).

In a sense, the pointwise character of services in standard SOA is replaced in our framework by a notion of service as the effect, triggered by some agents, of the overall ecosystem activity (including, e.g., the bonding together of some LSAs, the triggering of eco-laws that act on existing LSAs, the effect of other agents that are consequently activated, and so on).

4. A Concrete Implementation

Having introduced the proposed abstract architecture, we now analyze how it can be supported by existing concrete technologies. In particular, we will show that some technologies in the area of the Semantic Web community solidly match our problem domain. Here we first briefly overview our chosen technologies and their role in supporting the proposed architecture, and discuss perceived advantages over other candidate technologies (Section 4.1), describe details of implementation of LSAs into RDF (Section

4.2), show that the reasoning capabilities inherent to RDF provide one approach to realize the concept of semantic matching (Section 4.3), and describe details of how eco-laws can be expressed into SPARQL (Section 4.4).

4.1 The Resource Description Framework

The Resource Description Framework (RDF) (Miller and Manola, 2004) provides a formalization of a directed graph (with nodes representing resources and arcs representing properties). An RDF graph can be deconstructed to a set of *triples* $\langle s, p, o \rangle$, each asserting a relation, p, that holds between a subject, s and object, o. Subjects and relations in an RDF model are always resources (namely URIs[1], *namespace : string*) while objects may take the form of a resource or a literal value (a quoted string, possibly qualified by associated type information, e.g., using XML Schema datatypes).

RDF semantics are prescribed by two additional vocabularies, RDF Schema (RDFS) (Guha and Brickley, 2004) and the Web Ontology Language (OWL) (Krötzsch et al., 2009). RDFS provides a basic vocabulary for dividing RDF resources into classes, restricting the classes of resource a property may legally relate, and introduces *subClass* and *subProperty* properties to capture relations between classes and properties at different levels of abstraction. OWL provides a more expressive ontology language by, for example, supporting the expression of functional, transitive, symmetric, and inverse properties. Equivalent properties and classes may be declared, and cardinality restrictions allow constraints to be placed on the legal structure of class members. Off-the-shelf reasoners, e.g., Pellet (Sirin et al., 2007), can check the consistency of RDF models that use published vocabularies even if only partial data or partial ontology is available; this is desirable for a distributed scenario.

RDF is, in essence, a relational model for knowledge representation, which we use to model LSAs. We justify its selection over competing technologies by its two main advantages: domain-neutrality and natural support for data-distribution. RDF's domain-neutrality affords concurrent support for multiple applications across multiple domains, while its distributed data model supports seamlessly merging data from heterogeneous, distributed sources. Both benefits are a direct consequence of RDF's use of URIs to identify resources (i.e., giving triples unambiguous global semantics). This can be contrasted with, say, traditional database schemata, whose terms and relations have no prescribed semantics, and XML Schema, which is concerned with the structure of a data hierarchy and not with capturing the relations between data elements. In addition, neither technology is predisposed to integrating data adhering to multiple schemata.

The RDFS vocabulary sufficiently models the LSA component of our architecture. In this respect, we consider OWL an added value technology that supports the expression of semantics compatible with but not core to our approach. We explore one such extension, semantic matching, in Section 4.3.

Suitable technologies for inspecting and updating RDF stores exist in the form of SPARQL (Seaborne and Harris, 2009) and SPARQL Update (Gearon and Schenk, 2009). SPARQL supports queries consisting of triple patterns, conjunctions, negations, disjunctions, and optional patterns, while SPARQL Update supports the conditional insertion and removal of triples from an RDF store. SPARQL provides a means of describing to which LSAs an eco-law should apply, while SPARQL Update supports the expression of manipulations those LSAs are subject to—in the following, when there is no risk of ambiguity, we shall refer to the SPARQL *framework* to mean these two languages and related tools.

[1] Anonymous resources (sometimes referred to as blank nodes) designated with a locally-scoped identifier are also permitted by RDF, however we do not use them here.

RDF and the SPARQL framework, then, support the representation of distributed LSAs using terms from the most appropriate vocabularies for a particular environment, scenario, or set of applications, with the resultant data straightforwardly integrated, inspected, and updated using standard tooling. In particular, RDF stores can be used as spaces of LSAs, installed in each node of the pervasive system, maintaining the LSAs injected there in the form of groups of triples. Query engines, supporting execution of SPARQL queries and updates, and reasoners (Sirin et al., 2007) (coupled with simple network agents providing relocation of LSAs) can be used to support scheduling and execution of eco-laws locally—and in particular, to support advanced matching abilities as described in the following sections.

4.2 Serialization of LSAs

The main principle behind the idea of using RDF to represent an LSA is to construct an RDF "packet", featuring a set of triples $\langle i, p, v\rangle$ with same subject *i*, the LSA-id, and a pair *p*, *v* per each assignment of a property *p* to a distinct value *v*. Equivalently, an LSA is hence modeled as an identifier *i* followed by an unordered list of multi-valued properties—namely, a semantic tuple. One such value could be the identifier of another LSA, by which we model bonds. Accordingly, we are able to represent LSA-identifiers as possible subjects (and objects) for RDF triples, properties as predicates (and objects), and values of primitive datatypes (numbers, strings, and so on) as objects, namely, serializing LSA-ids and properties as URIs, and by mapping primitive data types to appropriate literal values (using XML Schema datatypes where appropriate). To illustrate, the RDF snippet (represented in Notation3 syntax (Berners-Lee and Connolly, 2011)) corresponding to the LSA of a user wandering the museum could be:

```
ex:lsa1432
  museum:type museum:person ;
  museum:location loc:room131 ;
  museum:time "2011-05-31T12:45:39"^^xsd:dateTime ;
  foaf:age "20"^^xsd:integer ;
  ex:interest "music" ;
  ex:interest "sport" ;
  ex:interest "travelling" ;
```

4.3 Semantic Matching

In addition to RDF's benefits as a data representation and exchange technology, the accompanying semantics provided by RDFS and OWL 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 by off-the-shelf reasoners, for example Pellet (Sirin et al., 2007), and inferences – such as the classification of resources – may be drawn. Indeed, standard OWL classification provides one approach to realizing semantic matching in the eco-law language. When two LSAs have to be matched, one can used relation <`?A rdf:type ?B`> using OWL semantics, which seeks for two LSAs – with id `?A` and `?B` – such that description of `?A` satisfies the set of restrictions that describe `?B`, an OWL Class description.

To illustrate, we discuss an example of a user in a shopping mall, interested in blue desks, 1 meter in height. In N3 notation, this description is constructed as:

```
:ProductDescription
  rdf:type owl:Class ;
  owl:equivalentClass
  [ rdf:type owl:Class ;
    owl:intersectionOf (
      [ rdf:type owl:Restriction ; owl:onProperty :height ; owl:hasValue :1METRE]
      [ rdf:type owl:Restriction ; owl:onProperty :productType ; owl:hasValue :Desk]
      [ rdf:type owl:Restriction ; owl:onProperty :colour ; owl:onClass :Blue ;
        owl:minQualifiedCardinality "1" ]
    )
  ] .
```

Consider also part of an LSA describing a desk's features of the kind

```
:ExampleDesk
  rdf:type :ProductDescription ;
  :height :100CM ;
  :productType :Desk ;
  :colour :MidnightBlue .
```

that differs in color and in its expression of height from the request. Assuming the vocabularies that define these terms also encode the knowledge that `ex:MidnightBlue` is of type `ex:Blue`, and that the term `ex:1METRE` is equivalent to `ex:100CM`, a reasoner can infer that the desk satisfies the description of Bob's interest.

This generalizes to other relations that a reasoner may dynamically compute. For example, the body of work relating to the semantic matching of web services defines many categories of match, including: *exact*, *plug-in*, *subsumes*, *intersection*, and *syntactic* (Bandara et al., 2008). Substituting `rdf:type` with terms from a vocabulary that describes these concepts, and introducing functions to compute such relations, supports their seamless integration into eco-laws.

4.4 The Eco-law Language, and its SPARQL Serialization

Eco-laws can be structured as chemical-resembling rules working on patterns of LSAs: they work by consuming a set of *reactant* LSAs based on left-hand side patterns and produce a set of *product* LSAs based on right-hand side patterns. They also obey a numeric transformation rate *r* written in the reaction arrow and representing a Markovian rate in a continuous-time Markov chain (CTMC) system. When not specified, this defaults to ∞, meaning the reaction is immediate. The rate influences the scheduling policies—an eco-law with rate *r* is scheduled after an elapsed time following negative exponential distribution of probability with average time *r* time units.

A pattern is used to match an annotation, and is formed by a variable `?x` that will hold the id of the annotation, and a sequence of filters used to control which annotations match—a pattern with an empty sequence of filters `?x:[]` is shortened to `?x`.

Filter `clones ?x.D(/extends ?x.D)` matches LSAs whose description exactly has(/includes) the property-value assignments of `x`—subsequent filters are then applied. Other filters constrain the values associated to a property in a point-wise manner: they take a term property on the left and a term property on the right. Case `p = (v1,..,vn)` matches those annotations in which property `p` is assigned *precisely* to values `v1,..,vn`, case `p` has `(v1,..,vn)` when `p` is assigned *at least* to values `v1,..,vn`, and `p` has-not `(v1,..,vn)` when `p` is assigned to *no* values in `v1,..,vn`. Variables identify properties on the left of an operator or lists of values/variables on the right. Besides the unconstrained variable `?x`, which can match any value, we also allow syntax `?{x:f}`, which can match any value `v` such that substituting `?x` with `v` in formula `f` gives a Boolean expression evaluating to true. Formulas are generated out of terms (values and variables) using application-dependent or standard mathematical unary and binary operators.

Once LSAs are turned into RDF, it is natural to try to consider existing languages to query and manipulate RDF stores, like SPARQL, as possible target for the eco-law language. In particular, the premise of our translation is to convert an eco-law into two fragments: *(i)* a SPARQL query (or simply a *query*) playing the role of the eco-law reactant patterns, namely, checking for the existence of reactant LSAs (finding instantiations for all the variables in the left-hand side); *(ii)* a sequence of SPARQL Update statements (or simply *statements*), obtained by instantiating all variables bound in the previous SPARQL query, whose final effect is to update LSAs as prescribed by the eco-law.

Technically, we concretize eco-law variables as query variables of the form `?x`, and write formulas *f* in the language of SPARQL FILTER and/or BIND constructs. We also introduce special notation (i.e., a parameter) `!x` in statements which is not part of SPARQL Update: the idea is that – before the statement is executed – `!x` is substituted with the value bound to former query variable `?x`. Figure 3 provides two example translations, each composed of a query and a sequence of statements, for two eco-laws [YOUNGEST] and [BOND-PV]. Eco-law [YOUNGEST] is used to aggregate two LSAs, with id `?FIELD` and `?FIELD2`. The former is a cloned copy of an LSA source `?L`, created at time `?T` and then diffused in the network, the latter is a clone of the former, created at greater time `?T2` and later diffused at a different time `?DT`: as a result, we only take `?FIELD` (namely, `?FIELD2` is removed) though its content is cloned from that of `?FIELD2`—namely, the resulting LSA retains the LSA-id of the older and content of the younger (such that new information overwrites old). Eco-law [BOND-PV] is used to create a bond from a source that specifies a property/value assignment that the target must include. It takes a `?SOURCE` that is not already bound to a `?TARGET` by property `?B`, and that specifies via bond `request` an LSA of type `request_pv` describing that a bond has to be created using `?B` towards an LSA with property `?PROP` featuring values `?VALUES`. The only result of applying this eco-law is that – after/if one such LSA `?TARGET` is found – `?SOURCE` is bonded to `?TARGET` by property `?B`.

An eco-law is first translated into a single initial `SELECT` query, where each operation filter that occurs in the reactant pattern is turned into a `WHERE` clause that checks the necessary conditions. For instance, in [BOND-PV], line 2 handles "`bond:request has (?BOND-REQ)`" (triple `?SRC bond:request ?BOND-REQ` should occur in the RDF store), line 3 handles "`?B has-not (?TARGET)`" (the triple should not exist), lines 4,5 handle "`?PROP has ?VALUES`" (there should be no value `o` assigned to `bond:target_value` in `?BOND-REQ` that is not assigned to `?PROP` in `?TARGET`), line 6 handles "`sapere:type has bond:request_pv`" (similar to line 2), lines 7, 8, 9 handle "`bond:bond_prop=(?B)`" (the triple should exist but no other value should occur for the same property), and similarly lines 10, 11, 12—filter "`bond:target_value=?VALUE`" is already dealt with in lines 4, 5. Only one `INSERT` statement is needed, to add one RDF triple to the store.

The case of [YOUNGEST] is similar, but exposes tricky aspects related to the clones construct and filters. Line 7 shows that annotated variables generate additional filters, lines 13 − 18 handle cloning in the left-hand side (except for properties `pump:pump_time` and `pump:diff_time`, which are mentioned in filters, any triple with subject `?FIELD` should exist in `?FIELD2` and vice-versa). Regarding update statements, triples where `?FIELD2` is the subject are first cleared and then reconstructed using those involving `?FIELD`.

The full translation of eco-laws language into the SPARQL and SPARQL UPDATE dialects is not reported here for brevity.

Eco-law [YOUNGEST]

```
?FIELD:[pump:source=(?L); pump:pump_time=(?T)]  +
?FIELD2:[clones ?FIELD.D; pump:pump_time=(?{T2: ?T2 > ?T}); pump:diff_time=(?DT)]
--->
?FIELD:[clones ?FIELD2.D]
```

Translation of [YOUNGEST]

```
SELECT DISTINCT * WHERE{
  ?FIELD pump:source ?L .
  FILTER NOT EXISTS {?FIELD pump:source ?o . FILTER (?o!=?L)}
  ?FIELD pump:pump_time ?T .
  FILTER NOT EXISTS {?FIELD pump:pump_time ?o .
                     FILTER (?o!=?T)}
  ?FIELD2 pump:pump_time ?T2 . FILTER (?T2 > ?T).
  FILTER NOT EXISTS {?FIELD2 pump:pump_time ?o .
                     FILTER (?o!=?T2)}
  ?FIELD2 pump:diff_time ?DT .
  FILTER NOT EXISTS {?FIELD2 pump:diff_time ?o .
                     FILTER (?o!=?DT)}
  FILTER NOT EXISTS {?FIELD2 ?p ?o .
      FILTER(?p!=pump:pump_time) FILTER(?p!=pump:diff_time)
      FILTER NOT EXISTS{?FIELD ?p ?o}}
  FILTER NOT EXISTS {?FIELD ?p ?o .
      FILTER(?p!=pump:pump_time) FILTER(?p!=pump:diff_time)
      FILTER NOT EXISTS{?FIELD2 ?p ?o}}
}
REMOVE {!FIELD2 ?p ?o} WHERE {!FIELD2 ?p ?o}
INSERT {!FIELD2 ?p ?o} WHERE {!FIELD ?p ?o}
```

Eco-law [BOND-PV]

```
?TARGET:[?PROP has ?VALUES]  +
?SRC:[bond:request has (?BOND-REQ); ?B has-not (?TARGET)]   +
?BOND-REQ:[sapere:type has (bond:request_pv); bond:bond_prop=(?B);
           bond:target_prop=(?PROP); bond:target_value=?VALUES]
--->
?SRC:[?B has (?TARGET)]  +  ?BOND-REQ  +  ?TARGET
```

Translation of [BOND-PV]

```
SELECT DISTINCT * WHERE{
  ?SRC bond:request ?BOND-REQ .
  FILTER NOT EXISTS {?SRC ?B ?TARGET}
  FILTER NOT EXISTS {?TARGET ?PROP ?o .
      FILTER NOT EXISTS {?BOND-REQ bond:target_value ?o}}
  ?BOND-REQ sapere:type bond:request_pv .
  ?BOND-REQ bond:bond_prop ?B .
  FILTER NOT EXISTS { ?BOND-REQ bond:bond_prop ?o.
                      FILTER (?o!=?B) }
  ?BOND-REQ bond:target_prop ?PROP .
  FILTER NOT EXISTS { ?BOND-REQ bond:target_prop ?o.
                      FILTER (?o!=?PROP) }
}
INSERT DATA {!SRC !B !TARGET .}
```

Figure 3: Example translations: (top) [YOUNGEST] eco-law – of two fields with the same source, it retains ID of the older and content of the younger; (bottom) [BOND-PV] eco-law – bond based on a property-value combination on target.

5. Adaptive Displays for Crowd Steering Application

We consider a crowd steering scenario as a case study to exemplify the approach and show how it tackles some of the requirements discussed, namely, situatedness and adaptation. In this example we guide people towards events of interest in a complex, dynamic environment (semantically matching people's interests), avoiding obstacles such as crowded regions, and without any supervised approach, namely, in a self-organizing 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 (called sensor nodes). These devices exchange information based on proximity, sense the presence of visitors, and hold information of various kinds of LSAs (e.g., about exhibits currently active in the museum). Visitors are equipped with a smartphone device that holds their preferences. By interacting with sensor nodes, visitors can be guided towards rooms with targets matching their interests, via signs dynamically appearing on the smartphone.

5.1 General Eco-laws

The only means by which an agent can get contextualised, such that it can gather information and decide how to act, is by the bonding mechanism—which allows an agent to read an LSA it did not inject itself. The agent typically cannot create those bonds, because it simply does not know which other LSAs are around – at least, initially – and which are pertinent. The typical situation is instead that some eco-law combines an agent's LSA with related ones forming its context, and accordingly creates a bond. Either such eco-laws act in an application-specific way based on the structure of the two LSAs, or some general mechanism has to be devised to let the agent manifest the intention of being bonded with LSAs having certain characteristics. One possible approach to bonding is shown by eco-law [BOND-EXT] presented in Figure 4, which combines a *source* LSA – exposing the description of a *target* – with the target itself, creating the bond. The owning agent can destroy bonds manually when there is need of doing so.

Such an eco-law assumes that the source defines a property `bond:request_ext` storing a bond towards an LSAs that describes all the aspects of the bonding request, which includes a reference to the source's property that will store the bond (`bond:bond_prop`), and most importantly a description of which LSAs can be a good target. The latter is in the form of another LSA (whose id is `?CONTENT`) which the target should extend, namely, `?CONTENT` defines all the property-value associations that the target should include.

The other eco-laws in Figure 4 are used to establish so-called computational fields (Mamei and Zambonelli, 2009; Viroli et al., 2011b; Beal and Bachrach, 2006; Viroli et al., 2011a; Pianini et al., 2011), a remarkable self-organization mechanism. A computational field is a data-structure distributed in a networked system based on spatial abstractions (distances, regions, and so on). Examples of computational fields include: *gradients*, mapping each node to the minimum distance from a source; *paths*, mapping each node belonging to the optimal path from a source and a distance to a non-zero value; *partitions*, mapping each node to the nearer in a finite set of nodes; and so on—see the Proto initiative (MIT Proto, 2010; Bachrach et al., 2010). The gradient case, for instance, is a very important one in pervasive computing systems, for it makes a possibly large set of nodes that surround a single one (the gradient source) aware of its state (or parts of it), and aware also of how it can be reached efficiently (i.e., along an optimal path)—hence supporting long-distance interactions.

By eco-law [PUMP], a gradient data structure is eventually established out of one LSA (`?SOURCE`) featuring non-empty properties `pump:req_field_with_range` (storing the gradient horizon), `pump:pump_rate` (the gradient pumping rate), and `pump:slope` (the propagation dynamics). Altogether those eco-laws diffuse some *field* LSAs around, converging to a situation in which: *(i)* each node within the `pump:range_distance_from_the_source` (computed by the estimated distance between nodes or on a step-by-step basis depending on the chosen slope) will carry one LSA of type

Universal Eco-laws

```
[BOND-EXT]: Bond based on the target extending a whole description
?TARGET:[extends ?CONTENT.D]  +
?SRC:[bond:request has (?BOND-REQ)]   +
?BOND-REQ:[sapere:type has (bond:request_ext); bond:bond_prop=(?B); bond:content=(?CONTENT)]
--->
?SRC:[?B has (?TARGET)]  +  ?BOND-REQ  +  ?TARGET

[PUMP]: Creates a field from an LSA
?LSA:[pump:req_field_with_range=(?RNG); sapere:location=(?L); pump:pump_rate=(?RATE)]  +
?TIME:[sapere:type has (sapere:time); sapere:time=(?T)]
--?RATE-->
?LSA  +  ?TIME  +
?SRC:[extends ?LSA.D; pump:range=(?RNG); pump:pump_time=(?T); pump:distance=("0");
      sapere:type has (pump:field, pump:source); pump:req_field_with_range has-not (?RNG);
      pump:source=(?LSA); pump:source_loc=(?L); pump:prev_ann=(?LSA); pump:prev_loc=(?L)]

[DIFF]: A field diffuses a cloned version in a neighbouring space
?FIELD:[sapere:type has (pump:field); pump:distance=(?D); pump:slope=(pump:distance);
        sapere:location=(?L); pump:range=(?R); pump:diff_rate=(?RATE); pump:follow=()] +
?NEIGH:[sapere:type has (sapere:neighbour); sapere:neighbour_location=(?L1),
        pump:distance=(?{D1: ?D1<?R-?D})]
?TIME: [sapere:type has (sapere:time); sapere:time=(?T)]
--?RATE-->
?FIELD + ?NEIGH + ?TIME +
?CLONE:[clones ?FIELD.D; pump:distance=(?{D2: D2 is ?D+?D1}), pump:prev_ann=(?FIELD);
        pump:prev_loc=(?L); sapere:location=(?L1); pump:diff_time=(?T);
        sapere:type has-not (pump:source,pump:field); sapere:type has (pump:prefield)]

[PRE]: A pre-field becomes a field if it contextualisation property  is set
?PREF:[sapere:type has (pump:pre-field); pump:contextualising=()]
--->
?PREF:[sapere:type has-not (pump:pre-field); sapere:type has (pump:field)]

[NEWEST]: Of two fields, we keep the one with newest information
?FIELD:[sapere:type has (pump:field); pump:source=(?L); pump:pump_time=(?PT)]   +
?FIELD2:[sapere:type has (pump:field); pump:source=(?L);
         pump:pump_time=(?{PT2: ?PT2 < ?PT})]
--->
?FIELD

[SHORTEST]: Of two fields pumped at same time, keep one with shorter distance
?FIELD:[sapere:type has (pump:field); pump:source=(?L); pump:pump_time=(?PT);
        pump:distance=(?D)]  +
?FIELD2:[sapere:type has (pump:field); pump:source=(?L); pump:pump_time=(?PT);
         pump:distance=(?{D2: ?D2 > ?D})]
--->
?FIELD
```

Figure 4: Eco-laws for universal generation of gradient data structures.

`sapere:field` that clones the source; *(ii)* such an LSA has an updated `pump:distance` property (by following decreasing values of distance any agent can retrieve an optimal path towards the source); *(iii)* the overall structure is able to self-organize (namely, self-heal) to topological changes (Beal, 2009; Mamei and Zambonelli, 2009).

Eco-law [DIFF] continuously diffuses a field LSA in neighboring locations (one at a time), at rate `pump:diff_rate`, with increasing distance value depending on the estimated distance to the neighbor, and providing the overall distance does not escape the gradient range. Diffusion is realized by creating LSAs whose property `sapere:location` stores the identifier of a neighboring node: an underlying middleware service is in charge of relocating LSAs according to the content of this property.

Additionally, the contextual information about available neighboring nodes is reified by the middleware in terms of LSAs with property `sapere:type` set to `sapere:neighbour`, by which relocation is ultimately specified. Note that the diffused LSA is actually of type `sapere:pre-field`, for it might need to be manipulated before being stored into one of type `sapere:field`.

Eco-law [PRE] implements the default manipulation behavior (activated if `pump:contextualizing` is empty): it stores the diffused LSA as it is—it needs an application specific eco-law to change this behavior, as we will develop next. Note that [PUMP, DIFF] keep creating new LSAs over time in the diffusion region; to remedy the inevitable divergence, [NEWEST] keeps the more recent field information in a node, while [SHORTEST] keep information of field LSAs with smaller distance from source.

5.2 LSAs

According to the proposed framework, all information exchanged, including agent representations, is encapsulated by LSAs, such as: (*i* - *source* LSAs) representing items or exhibits 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 intermediates to establish the final gradient; (*iv* - *user* LSAs) representing presence and state of a user (stored in their smartphone); (*v* - *crowd* LSAs) representing the presence of a crowded area by a sensor node. In particular, we rely on source LSAs of the kind

```
museum:sourcelsa1432
  sapere:type  (museum:exhibition) ;
  sapere:location (loc:room131) ;
  sapere:time ("2011-05-31T12:45:39"^^xsd:dateTime) ;
  pump:req_field_with_range ("1000"^^xsd:integer) ;
  pump:pump_rate ("1"^^xsd:integer) ; // once per minute
  pump:slope (pump:distance) ;        // proceeding based on distance
  pump:contextualising (pump:crowd) ;
  pump:decay_time ("60"^^xsd:integer) ; // in minutes
  museum:author (museum:michelangelo) ;
  museum:kind (museum:sculpture)
```

describing an exposition of Michelangelo's sculpture. A crowd LSA, generated in each sensor node takes the form:

```
museum:crowdlsa1123
  sapere:type (museum:crowd) ;
  sapere:location (loc:sensor1231) ;
  sapere:time ("2011-05-30T11:00:00"^^xsd:dateTime) ;
  museum:crowdlevel ("1.0"^^xsd:decimal) ;
  museum:crowdfactor ("10.0"^^xsd:decimal)
```

where `museum:crowdlevel` set to 1.0 means that the sensor perceived the highest crowd, while crowd factor is a multiplication factor dictating how such a level should penalize a gradient. Finally, the LSAs of a user – seeking (field) exhibitions of any Renaissance's sculptor – is of the kind:

```
museum:personlsa81
  sapere:type (museum:person) ;
  sapere:location (loc:room112) ;
  sapere:time ("2011-05-31T12:21:39"^^xsd:dateTime) ;
  bond:request (museum:req1211) ;
  user:profile (user:content99)

museum:req1211
  sapere:type (bond:request_wm) ;
  bond:bond_prop (museum:exhibitions) ;
  bond:local_prop (user:profile)

user:content99
  sapere:type (museum:exhibition) ;
  sapere:type (pump:field) ;
  sapere:author (museum:renaissance_sculptor)
```

Note that this LSA will bond to field LSAs generated by `museum:sourcelsa1432` above thanks to eco-law [BOND-WM]. Also, by the structure of source LSA we can expect eco-laws [PUMP, DIFF-DST, YOUNGEST, SHORTEST, DECAY] to activate and support a stable gradient. However, a further application-dependent eco-law is needed since eco-law [PRE] is not effective here, for contextualisation is required to handle crowded areas. One such eco-law is:

```
[PRE-CROWD]: A pre-field becomes a field after increasing its distance due to crowd
?PREF:[sapere:type has (pump:pre-field, museum:exhibition);
       pump:contextualising=(museum:crowd); pump:distance=(?D)]  +
?CROWD:[sapere:type has (museum:crowd); museum:crowdlevel=(?L),
        museum:crowdfactor=(?F)]
--->
?PREF:[sapere:type has-not (pump:pre-field); sapere:type has (pump:field);
      pump:distance=?D2: ?D2 is ?D+?L*?F]  +
?CROWD
```

which increases the field LSA distance (i.e., a penalty) when the local crowd level is greater than 0.

When a user perceives exhibits and is bonded to field LSAs in its locality (by eco-law [BOND-EXT]), it will choose one such source to be steered to, and accordingly updates its LSA to:

```
museum:personlsa81
  sapere:type (museum:person) ;
  sapere:location (loc:room112) ;
  sapere:time ("2011-05-31T12:21:39"^^xsd:dateTime) ;
  museum:chosen_exhibition (museum:sourcelsa1432) ;
  bond:request (museum:req121)

museum:req1211
  sapere:type (bond:request_ext) ;
  bond:bond_prop (museum:field_to_follow) ;
  bond:content (museum:content100)

museum:content100
  sapere:type (museum:exhibition) ;
  sapere:type (pump:field) ;
  pump:source (museum:sourcelsa1432)
```

such that it will bond to the desired field, from which information on the directions to take can be observed and will be updated as it moves thanks to gradient automatic self-healing.

5.3 Simulation

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 is fixed, the evolution of a service ecosystem can be simulated using any available framework for CTMCs, like e.g., PRISM (University of Birmingham, 2007) (which also allows for stochastic model-checking), typically working via Stochastic Simulation Algorithms (SSA) based on (Gillespie, 1977).

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

A second set of tests was aimed at verifying the management of overcrowding and, in particular, how the behavior 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 quickly 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.

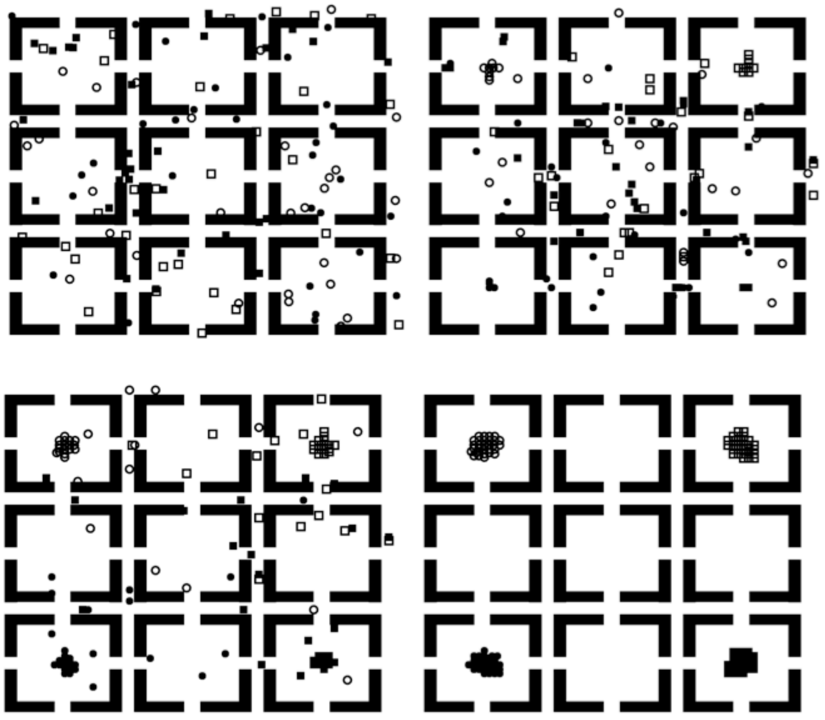

Figure 5: A simulation run of the reference exposition (top-left, top-right, bottom-left, bottom-right): from random positions people move to 4 targets such that it will bond to the desired field, from which information on the directions to take can be observed and will be updated as it moves thanks to gradient automatic self-healing.

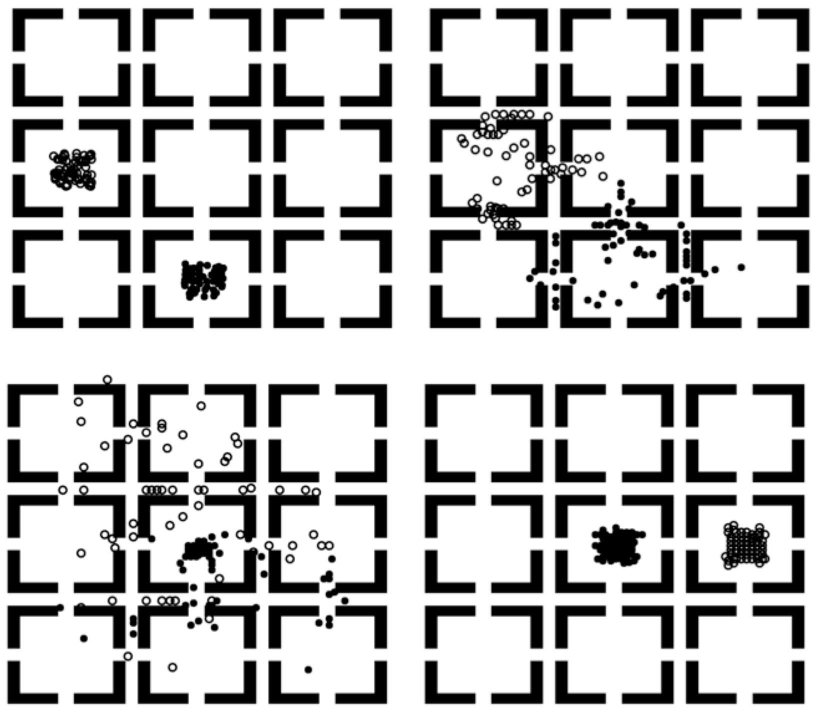

Figure 6: Dark visitors occupy a central room: others move left to right by a longer, less crowded path circumventing the central room on top.

Both tests show qualitative effectiveness of the proposed eco-laws, and suggest that our simulation approach can be used for additional experiments focusing on tuning system parameters (factor *k*) or alternative strategies (e.g., diffusing crowd information) to optimize paths to destinations.

6. Related Works

6.1 Traditional tuple spaces

Interaction of agents in our framework is mediated by a space reifying and ruling interaction. This very idea has been proposed by the pioneer work in the Linda (Gelernter, 1985) tuple space model and its distributed implementations (Noble and Zlateva, 2001; Nielsen and Sørensen, 1993; Merrick and Wood, 2000; Atkinson, 2008). In Linda – which originated as a language for parallel computing but was later adopted as paradigm for distributed system coordination (Omicini and Viroli, 2011) – coordination is achieved by coordinated processes inserting tuples (records of primitive values) in the global shared space and retrieving (and removing) them based on templates (records possibly including wildcards). Retrieval is always blocking, though non-blocking versions are typically available in the above implementations.

Despite the evident LSA/tuple and LSA-space/tuple-space similarity– tuple spaces were indeed a source of inspiration for our work – our framework introduces a profoundly different approach to the engineering of distributed systems. Tuple space approaches see tuples as (possibly structured) “tokens” to be placed/retrieved in the shared space either as a means of synchronization or communication. LSAs are instead strictly bound to their owner: they represent both its “interface” towards the environment (when considering manifestation) and the environment “interface” for the agent (when considering observation). In this acceptation, the concept of LSA resembles and develops on top of the notions of Agent Coordination Context as developed in (Omicini et. al., 2004) and coordination artifact (Omicini et. al., 2006).

6.2 Semantic Tuple Spaces

So-called *semantic tuple space computing* has been recognized as one of the most notable advances in the field of middlewares for dealing with issues like openness, dynamism, and scalability—that are peculiar aspects of applications operating in open environments such as the Web. It basically amounts to augmenting tuple spaces with semantic representation and reasoning—an overview of some approaches is in Nixon et al., 2008, which we here briefly review and complete.

Triple space computing (TSC) (Fensel, 2004) provides an extension of tuple space computing with features of Semantic Web technology. In particular, this work proposes an extension of the classical flat data model adopted for tuples: it relies on the use of RDF to support applicability in the Semantic Web domain. This model is based on the assignment of URIs to tuples, which can be interlinked so as to form graphs. Moreover, tuples becomes RDF triples of a subject, a predicate and an object. Among the others, one of the main advantages of adopting Semantic Web technology lies in the possibility to provide a tuple space with more refined (semantic) matching algorithms than the usual template matching of Linda.

Born as an extension of TSC, Conceptual Spaces (CSpaces) is an independent initiative targeted at studying the applicability of semantic tuple space computing in scenarios other than Web, such as Ubiquitous Computing, Distributed Software Components, Enterprise Application Integration and so on. CSpaces features several components: a knowledge container, an organizational and a coordination model, a model for semantic interoperability, a security and trust model, a knowledge visualization model, and an architecture model. In particular tuples are records with exactly 7 fields, one of which being a logical formula expressed e.g. in Description Logics and representing the semantic content of the tuple. Other fields roughly resembles our synthetic properties and identifiers—for a thorough description of these components, the interested reader can refer to (Nixon et al., 2008).

Semantic Web Spaces (Tolksdorf et al., 2008; Nixon et al., 2008) has been devised as a middleware for the Semantic Web, where clients can exploit Semantic Web data to access and manipulate knowledge, so as to coordinate their activities. In particular, originally conceived as an extension of XMLSpaces – a tuple space platform supporting the exchange of XML documents – Semantic Web Spaces provides tuple spaces able to support the exchange of RDF triples (i.e., tuples), relying on RDFS reasoning capabilities as regards to matching.

The sTuples model (Khushraj et al., 2004; Nixon et al., 2008), developed by Nokia Research Center, is targeted at pervasive computing settings. There, given the heterogeneous and dynamic nature of pervasive environments, the combined adoption of Semantic Web and tuple spaces have been recognized as a viable solution not only to semantic interoperability issues, but also with respect to temporal and spatial decoupling, and synchronization. To this end, semantic tuples (based on JavaSpace object tuples) are provided that extend data tuples, and tuple template matching is extended via a semantic matching mechanism on top of object-based matching. In addition, sTuples features agents residing in the space with the goal of performing user-centric services, such as tuple recommendation.

The RDFSwarms model in (Harasic et al., 2010) explores swarm approaches in semantic tuple spaces, combining self-organization aspects of tuple diffusion and manipulation with the semantic character of technologies like RDF, with the goal of optimizing the retrieval of tuples by semantic similarity.
A previous semantic tuple space approach of ours is presented in (Nardini et al., 2010), in which the TuCSoN tuple center model is extended with notions of semantic tuples and semantic templates, roughly corresponding to instance and classes in Description Logics, with a proposed tuple-like syntax used to retain simplicity of the approach. This work has been extended with concepts of fuzzy matching (Nardini et al., 2011).

Most of the above approaches limit tuples to just RDF triples, making it necessary to build complex graphs even to model simple individuals and concepts. One of the key features of our model, following the direction of (Nardini et al., 2010), is to consider a tuple as the overall description of an individual (e.g. as modeled in Description Logics), but inventing a language of tuples and of templates – namely, LSAs and eco-law patterns – that remains simple enough and does not expose the whole complexity of Description Logics—as e.g., in Conceptual Spaces. Differently from (Nardini et al., 2010), we address multi-valued properties and references to other individuals in a more natural and effective way, especially as far as translation to RDF is concerned.

6.3 Self-organization in Tuple Spaces

As described in (Omicini and Viroli, 2011), applications of coordination models and languages – and especially space-based ones – are inevitably entering the realm of self-organization, 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 organize and evolve tuples according to specified rules.

TOTA (Tuples On The Air) (Mamei and Zambonelli, 2009) 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 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).

Concerning TOTA, which is the model most similar to ours from the viewpoint of application domain, we observe that while our 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 behavioral 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: an agent defines diffusion behavior before injecting the tuple in the system. Embedding the behavior in the tuples, rather than in the space, make difficult and impractical the task of predicting overall ecosystem behavior in advance.

Finally, it is worth noting that the model presented here originates from previous work of ours (Viroli and Casadei, 2009; Viroli et al., 2011b; Viroli and Casadei, 2010), where a bio-chemical tuple space model is 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.

6.4 Other Semantic Rule-based Approaches

The eco-law language bears a resemblance to other rule languages based on Semantic Web technologies, i.e., those designed (or bridged) to adhere to and operate over the RDF model.

A first category includes human-friendly languages designed primarily for readability, and localized to a particular piece of software. For example, Jena's rule language and associated inference engine (Carroll et al., 2004) supports manually specified rules, and is used within Jena to infer knowledge entailed by RDFS and OWL semantics. TRIPLE (Decker et al., 2005) is a proposed query and rule language for RDF, while N3 Logic (Berners-lee et al., 2008) provides a framework that extends N3 RDF syntax with support for rules. Beyond languages designed specifically to operate over the RDF model, general-purpose rule engines, such as Jess (Friedman-Hill, 2008), and Prova (Paschke and Schroder, 2007), have been successfully bridged to operate over RDF (O'Connor and Knublauch, 2005; Paschke and Schroder, 2007).

A second category is that of rule markup languages, which are heavily tailored towards the publication, interchange, and reuse of rules across software systems. There sit the Semantic Web Rule Language (SWRL) (Horrocks et al., 2004), an RDF-based language that combines a subset of features of OWL and the Rule Markup Language (RuleML) (Boley et al., 2001), and the REWERSE Rule Markup Language (Wagner et al., 2006), which extends SWRL with support for Object Constraint Language (Warmer and Kleppe, 1999). SPARQL Inferencing Notation (SPIN) (Knublauch et al., 2011) provides a vocabulary to represent SPARQL queries as RDF, and supports the execution of rules and object constraints over RDF data models. Finally, the Rule Interchange Format (Kifer, 2008) is the product of an active W3C working group aiming to implement rule interchange between different rule languages in the Semantic Web.

Although a cosmetic correspondence between the structure of eco-laws and rules exists, differences in the goals and underlying semantics of each strongly differentiate them. In particular *(i)* eco-laws do not denote a logical inference, but a data transformation (that often modifies the law's reactants), *(ii)* working at the level of LSAs, eco-law operational semantics capture instructions to modify, create, delete, or connect "objects", rather that express the generation of new knowledge from the existence of combinations of individual "facts", *(iii)* the firing of an eco-law results in the transformation of a single pattern of LSAs matching the law's left-hand side; this can be juxtaposed with rule engines, where rule sets are usually evaluated as a batch, executing all matched patterns over all rules, finally *(iv)* eco-law scheduling is CMTC based, rather than priority based, which is typical of rule engines.

Points *(iii)* and *(iv)* additionally imply an entirely different methodology for working within the eco-law framework than with a rule engine. In a rule-based environment the rule set is typically evaluated immediately after new data is introduced or before the knowledge base is next queried, giving software agents the expectation that, *a)* the inference process with respect to the most recent input is complete, and *b)* in the absence of further input, the knowledge base remains static. Neither expectation holds within the eco-law framework, where software agents have no explicit control over the firing of eco-laws relative to the timing of its inspection of the environment, nor any expectation of knowledge stability within the LSA Space beyond the properties of LSAs that it directly governs.

Setting aside methodology and the semantics of execution, we note that parts of the eco-law semantics may be approximated by some of the rule languages described above. For example, a single LSA appearing on the left-hand side of an eco-law may be expressed in any rule languages as a conjunction of facts—one for each LSA property. Some rule languages (e.g., Jess) support the binding of new variables on the right-hand side of a rule, but this capability is not standard. The semantics of both the 'clones' and 'extends' operators are closely tied to the eco-law's "object oriented" view of the world—that is, both operate over knowledge that is implicitly declared by association to bound LSA identifiers. To the best of our knowledge, this precludes implementation of these operators in any the Semantic Web rule languages discussed above (e.g., Jena Rules and SWRL lack the capability to bind variables on the right-hand side of a rule, SWRL also does not support quantification necessary to operate over the set of properties associated with an LSA identifier). It appears possible to synthesize this functionality in the Jess language by using a combination of custom functions and procedural-style code within a rule, however this "algorithmic" approach to rule construction is non-standard.

Finally, we note that the existence of a formal translation from the eco-law language to SPARQL means that eco-laws may be executed against the majority of both open-source and proprietary RDF stores, using standard tooling.

7. CONCLUSION

In this chapter we presented a framework for pervasive service ecosystems, meant to overcome the limitations of standard SOA when dealing with situated and adaptive pervasive computing systems, along with an implementation schema based on standard W3C technologies for the Semantic Web, and an application case of crowd steering by public/private displays.

We believe that the main novelty of the proposed approach precisely lies in the interplay between eco-laws and LSAs: the former regulates the overall ecosystem by basic mechanisms of agent interaction via bonds and of spatial coordination (Viroli et al., 2011b); the latter regulates agent autonomy, controlling how a single agent is affected by the ecosystem and manifests to it. To this end, a key role is played by the liveness of LSAs, which is achieved by three fundamental mechanisms: *(i)* agents have responsibility for continuously updating the state of their LSAs over time; *(ii)* contextualization (seen as a middleware service) guarantees that information about the physical world promptly reflects in synthetic LSAs and properties; *(iii)* eco-laws enact continuous processes, evolving LSAs to reflect the overall ecosystem situation.

Concerning the use of W3C technologies, we remark that they come equipped with a set of reasoners and techniques we can adopt to devise analysis tools for pervasive ecosystems. The proposed translation also defines a reference implementation of an LSA-space based on an RDF-store coupled with a SPARQL interpreter. This does not mean that this is the preferred technological solution for the development of a middleware, but it provides a means to quickly prototype a correct-by-definition LSA-space implementation, and a reference to be compliant with in the case other solutions are considered. Finally, the approach described in this chapter paves the way for a deep connection with the use of existing/new OWL ontologies. They can be used not only to support OWL-based semantic matching, but also to declare the shape of valid LSAs, check/enact it, and enforce correctness of ecosystems developed on top of know LSAs and eco-laws.

Although necessarily early, we believe the experiments described in this chapter show some key properties of the proposed model: *(i)* expressiveness of the eco-law language in modeling basic interaction patterns up to sophisticated self-organization patterns; *(ii)* ability to compose universal and application-specific eco-laws; *(iii)* possibility of analyzing ecosystems – and the behavior of eco-laws therein – by simulation.

7.1 Future Works

The activities to be carried out in the next stages of this research include:

- To provide a formal operational model, grounding the proposed approach and giving semantics to the eco-law language.
- A methodology (or a coherent collection of methodology fragments) for the development of ecosystems will be developed, following early experiences in the context of the AOSE (agent-oriented software engineering) research field. In particular, the issue of application-independent versus application-specific eco-laws, and the problem of which application-independent eco-laws to provide to guide the basic aspects of ecosystem behavior, are to be studied in detail.
- Staying within the context of methodology, further exploration and deepening of the relationship between eco-law mediated interactions and traditional coordination technologies, with particular attention to BPEL, the business process model that describes interactions between web services.
- In order to work in a complete yet effective way, the methodology should be coupled with additional tools, including: *(i)* a fully-featured simulation framework for ecosystems, implementing in an efficient way the operational semantics of eco-laws; *(ii)* a workbench for the formal analysis of static and behavioral properties of ecosystems, possibly reusing existing tools tackling fragments of the eco-law language; *(iii)* implementing prototype development tools, including editors and checkers.

ACKNOWLEDGMENTS

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

REFERENCES

Atkinson, A. K. (2008). Tupleware: A distributed tuple space for cluster computing. *In Proceedings of the 2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 121– 126, Washington, DC, USA. IEEE Computer Society.

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

Bachrach, J., Beal, J., and McLurkin, J. (2010). Composable continuous space programs for robotic swarms. *Neural Computing and Applications*,19(6):825–847.

Bandara, A., Payne, T., Roure, D. D., Gibbins, N., and Lewis, T. (2008). Semantic resource matching for pervasive environments: The approach and its evaluation. Technical Report ECSTR-IAM08-001, School of Electronics and Computer Science, University of Southampton.

Beal, J. (2009). Flexible self-healing gradients. In Shin, S. Y. and Ossowski, S., editors, *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC)*, Honolulu, Hawaii, USA, March 9-12, 2009, pages 1197–1201. ACM.

Beal, J. and Bachrach, J. (2006). Infrastructure for engineered emergence on sensor/actuator networks. *IEEE Intelligent Systems*, 21(2):10–19.

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

Berners-lee, T., Connolly, D., Kagal, L., Scharf, Y., and Hendler, J. (2008). N3logic: A logical framework for the world wide web. *Theory and Practice of Logic Programming*, 8:249–269.

Boley, H., Tabet, S., and Wagner, G. (2001). Design rationale of RuleML: A markup language for semantic web rules. In *The Semantic Web Working Symposium*, pages 381–401.

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

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkin- son, K. (2004). Jena: implementing the semantic web recommendations. *In Proceedings of the 13th international World Wide Web conference - Alternate track papers & posters*, pages 74–83, New York, NY, USA. ACM.

Coleman, B. (2009). Using sensor inputs to affect virtual and real environments. *IEEE Pervasive Computing*, 8(3):16–23.

Decker, S., Sintek, M., Billig, A., Henze, N., Dolog, P., Nejdl, W., Harth, A., Leicher, A., Busse, S., Ambite, J. L., Weathers, M., Neumann, G., and Zdun, U. (2005). TRIPLE - an RDF rule language with context and use cases. *In Rule Languages for Interoperability*.

Fensel, D. (2004). Triple-space computing: Semantic web services based on persistent publication of information. In Aagesen, F. A., Anutariya, C., and Wuwongse, V., editors, *Intelligence in Communication Systems, IFIP International Conference*, *INTELLCOMM 2004*, Bangkok, Thailand, Proceedings, volume 3283 of Lecture Notes in Computer Science, pages 43–53. Springer.

Ferscha, A. and Vogl, S. (2010). Wearable displays – for everyone! *IEEE Pervasive Computing*, 9(1):7–10.

Friedman-Hill, E. (2008). Jess, the rule engine for the Java platform. http: //www.jessrules.com/.

Gearon, P. and Schenk, S eds. (2009). SPARQL 1.1 update. W3C working draft, W3C. http://www.w3.org/TR/2009/WD-sparql11-update-20091022/.

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

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361.

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

Harasic, M., Augustin, A., Obermeier, P., and Tolksdorf, R. (2010). RDFSwarms: selforganized distributed RDF triple store. *In Proceedings of the 2010 ACM Symposium on Applied Computing, SAC '10*, pages 1339–1340, New York, NY, USA. ACM.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). SWRL: A semantic web rule language combining OWL and RuleML. Technical report, W3C Member Submission. http://www.w3. org/Submission/SWRL/.

Jazayeri, M. (2005). Species evolve, individuals age. *In 8th IEEE International Workshop on Principles of Software Evolution*, pages 3–12, Washington, DC.

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

Khushraj, D., Lassila, O., and Finin, T. W. (2004). sTuples: Semantic tuple spaces. *In proceedings of the 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04),* pages 268–277, Boston, MA, USA.

Kifer, M. (2008). Rule interchange format: The framework. *In Web Reasoning and Rule Systems*, volume 5341 of Lecture Notes in Computer Science, pages 1–11. Springer Berlin / Heidelberg.

Knublauch, H., Hendler, J. A., and Idehen, K. (2011). SPIN: SPARQL inferencing notation. W3C member submission, W3C. http://www.w3.org/ Submission/2011/SUBM-spin-overview-20110222/.

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

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

Merrick, I. and Wood, A. (2000). Coordination with scopes. *In Proceedings of the 2000 ACM symposium on Applied computing - Volume 1*, SAC '00, pages 210–217, New York, NY, USA. ACM.

Miller, E. and Manola, F. eds. (2004). RDF primer. W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

MIT Proto (Retrieved Nov. 1st, 2010). MIT Proto. software available at http://proto.bbn.com/.

Nardini, E., Omicini, A., and Viroli, M. (2011). Description spaces with fuzziness. *In 26th Annual ACM Symposium on Applied Computing (SAC 2011)*, Tunghai University, TaiChung, Taiwan. ACM.

Nardini, E., Viroli, M., and Panzavolta, E. (2010). Coordination in open and dynamic environments with Tucson semantic tuple centres. *In proceedings of the 25th Annual ACM Symposium on Applied Computing (SAC 2010)*, volume III, pages 2037–2044, Sierre, Switzerland. ACM.

Nielsen, B. and Sørensen, T. (1993). *Distributed programming with multiple tuple space Linda*. Aalborg University. Institute for Electronic Systems.

Nixon, L. J. B., Simperl, E., Krummenacher, R., and Martin-recuerda, F. (2008). Tuplespace-based computing for the semantic web: A survey of the state-of-the-art. *The Knowledge Engineering Review*, 23(2):181–212.

Noble, M. S. and Zlateva, S. (2001). Scientific computation with Javaspaces. In *Proceedings of the 9th International Conference on High-Performance Computing and Networking, HPCN Europe 2001,* pages 657–666, London, UK. Springer-Verlag.

O'Connor, M. J. and Knublauch, H. (2005). Writing rules for the semantic web using SWRL and Jess. *In the Protege with Rules Workshop, held with 8th International Protege Conference*.

Omicini, A. and Ricci, A. and Viroli, M. (2005). RBAC for organisation and security in an agent coordination infrastructure. *In vol. 128 of Electronic Notes in Theoretical Computer Science, pages 65–85. Elsevier Science B.V.*

Omicini, A. and Ricci, A. and Viroli, M. (2006). Coordination artifacts as first-class abstractions for MAS engineering: State of the research. *In Software Engineering for Multi-Agent Systems, IV: Research Issues and Practical Applications, vol. 3914 di LNAI, pagg. 71–90. Springer*.

Omicini, A. and Viroli, M. (2011). Coordination models and languages: From parallel computing to self-organisation. *The Knowledge Engineering Review*, 26(1):53–59. Special Issue 01 (25th Anniversary Issue).

Paschke, A. and Schroder, M. (2007). Inductive logic programming for bio-informatics in Prova. *In proceedings of the 2nd Workshop on Data Mining in Bioinformatics at VLDB'07*.

Pianini, D., Montagna, S., and Viroli, M. (2011). A chemical inspired simulation framework for pervasive services ecosystems. *In Proceedings of the Federated Conference on Computer Science and Information Systems*, pages 675–682, Szczecin, Poland. IEEE Computer Society Press.

Seaborne, A. and Harris, S. eds. (2009). SPARQL 1.1 query. W3C working draft, W3C. http://www.w3.org/TR/2009/WD-sparql11-query-20091022/.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Web Semantics*, 5:51–53.

Spohrer, J. C., Maglio, P. P., Bailey, J. H., and Gruhl, D. (2007). Steps toward a science of service systems. *IEEE Computer*, 40(1):71–77.

Tolksdorf, R., Nixon, L. J. B., and Simperl, E. P. B. (2008). Towards a tuplespace-based middleware for the Semantic Web. *Web Intelligence and Agent Systems*, 6(3):235–251.

Ulieru, M. and Grobbelaar, S. (June 2007). Engineering industrial ecosystems in a networked world. *In proceedings of the 5th IEEE International Conference on Industrial Informatics, pages 1–7*. IEEE Press.

University of Birmingham (2007). The PRISM probabilistic model checker. Available at http://www.prismmodelchecker.org.

Vargo, S. L., Maglio, P. P., and Akaka, M. A. (2008). On value and value co-creation: a service systems and service logic perspective. *European Management Journal*, 26(3):145–152.

Viroli, M., Beal, J., and Casadei, M. (2011a). Core operational semantics of Proto. *In proceedings of the 26th Annual ACM Symposium on Applied Computing (SAC 2011)*, *volume II: Artificial Intelligence & Agents, Information Systems, and Software Development*, pages 1325–1332,

Viroli, M. and Casadei, M. (2009). Biochemical tuple spaces for self-organising coordination. *In proceedings of the 11th International Conference on Coordination Languages and Models*, volume 5521 of LNCS, pages 143–162. Springer, Lisbon, Portugal.

Viroli, M. and Casadei, M. (2010). Chemical-inspired self-composition of competing services. *In proceedings of the 25th Annual ACM Symposium on Applied Computing (SAC 2010), volume III,* pages 2029–2036, Sierre, Switzerland. ACM.

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

Wagner, G., Giurca, A., and Lukichev, S. (2006). A usable interchange format for rich syntax rules integrating OCL, RuleML and SWRL. *In Proceedings of the Workshop on Reasoning on the Web (RoW2006).*

Warmer, J. and Kleppe, A. (1999*). The object constraint language: precise modeling with UML*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Wells, G. C., Mueller, B., and Schulé, L. (2008). A tuple space web service for distributed programming - simplifying distributed web services applications. In proceedings of the Fourth International Conference on Web Information Systems and Technologies (WEBIST 2008), pages 93–100. INSTICC Press.

ADDITIONAL READING SECTION

Agha, G. (2008). Computing in pervasive cyberspace. *Communications of the ACM*, 51(1):68–70.

Androutsellis-Theotokis, S. and Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371.

Banâtre, J.-P. and Le Métayer, D. (1993). Programming by multiset transformation. *Communications of the ACM*, 36(1):98–111.

Barros, A. P. and Dumas, M. (2006). The rise of web service ecosystems. *IT Professional,* 8(5):31–37.

Corkill, D. (1991). Blackboard systems. Journal of AI Expert, 9(6):40–47.

Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., and Zambonelli, F. (2006). A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems*, 1(2):223–259.

Gardelli, L., Viroli, M., Casadei, M., and Omicini, A. (2008). Designing self-organising environments with agents and artefacts: A simulation-driven approach. *International Journal of Agent-Oriented Software Engineering*, 2(2):171–195.

Gelernter, D. and Carriero, N. (1992). Coordination languages and their significance. *Communications of* *the ACM*, 35(2):97–107.

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

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

Omicini, A., Ricci, A., and Viroli, M. (2008). Artifacts in the A&A meta- model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3).

Prigogine, I. and Steingers, I. (1997). *The End of Certainty: Time, Chaos, and the New Laws of Nature*. Free Press.

Viroli, M., Holvoet, T., Ricci, A., Schelfthout, K., and Zambonelli, F. (2007). Infrastructures for the environment of multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):49–60.

Viroli, M. and Zambonelli, F. (2010). A biochemical approach to adaptive service ecosystems. *Information Sciences*, 180(10):1876–1892.

Wegner, P. (1997). Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91.

Zambonelli, F. and Parunak, H. V. D. (2002). From design to intention: signs of a revolution. *In the first international joint conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*, pages 455–456. ACM.

KEY TERMS & DEFINITIONS

Adaptivity: possessing the ability to change behavior based on observed stimuli.

Autonomic: a characteristic of a computational resource or set of computational resources that can automatically adapt to certain types of environmental change without user intervention.

Eco-law: a chemical-like global coordination rule that both enables and regulates interactions between agents within a pervasive ecosystem.

Live Semantic Annotation: a continuously updated, structured, formal representation of a portion of an agent's state, intended to support coordination.

Pervasive ecosystem: the environment that results from the increased deployment and embedding of sensing technologies within everyday objects, designed to support general-purpose services that are based on opportunistic encounters between both static and highly mobile components.

Prosumer: a user, or agent that can potentially both contribute information to and consume information from their operating environment.

Situatedness: a property of a computation, action, or behavior that is influenced by the current state of the surrounding physical and social environment.

Spatial substrate: a middleware-constructed, logical partition of space that abstracts from the physical network structure and is designed to support situated interactions between agents.