Engineering Pervasive Multiagent Systems in SAPERE

Ambra Molesini[1], Andrea Omicini[1], Mirko Viroli[1], and Franco Zambonelli[2]

[1] Dipartimento di Informatica–Scienza e Ingegneria (DISI)
Alma Mater Studiorum–Università di Bologna, Italy
{ambra.molesini,andrea.omicini,mirko.viroli}@unibo.it
[2] Dipartimento di Scienze e Metodi dell'Ingegneria (DISMI)
Università degli Studi di Modena e Reggio Emilia, Italy
franco.zambonelli@unimore.it

Abstract Given the growth of agent-based models and technologies in the last decade, nowadays the applicability of agent-oriented techniques to the engineering of complex systems such as pervasive computing ones critically depends on the availability and effectiveness of agent-oriented *methodologies*. Accordingly, in this paper we take SAPERE pervasive service ecosystems as a reference, and introduce a novel agent-oriented approach aimed at engineering SAPERE systems as multi-agent systems.

1 Introduction

The ICT landscape has dramatically changed with the advent of mobile and pervasive computing technologies. The dense spread in our everyday environment of sensor networks, RFID tags, along with the mass diffusion of always-on-line smart phones and mobile social networking, is contributing to shape an integrated infrastructure that can be used for the provisioning of innovative general-purpose digital services [1,2]. In particular, such infrastructure will be used to ubiquitously access services improving interaction with the surrounding physical world as well as the social activities therein. Users will be expectedly able to deploy customised services, making the overall infrastructure as open as the Web currently is [3].

According to the above trends, a great deal of research activity in pervasive computing and service systems has been recently devoted to solve problems associated to the design and development of effective pervasive service systems. They include: supporting self-configuration and context-aware spontaneous composition; enforcing context-awareness and self-adaptability; and ensuring that service frameworks can be highly-adaptive and very long-lasting [4]. Unfortunately, most of the solutions so far are proposed in terms of "add-ons" to be integrated in existing frameworks [5,6,7]. The result is often an increased complexity of current frameworks and, in the end, a lack of clean and usable methodological approaches to the engineering of complex pervasive services systems.

Against this background, here we elaborate on the SAPERE novel approach to the engineering of complex pervasive service system [8]. SAPERE (short for

"Self-Aware PERvasive service Ecosystems") tackles the problem of engineering distributed pervasive service systems by a foundational re-thinking of distributed systems, i.e., grounding on a nature-inspired [9,10], and specifically bio-chemically inspired approach, to effectively support context-awareness, spontaneous service composition, and self-adaptivity. Specifically, SAPERE attacks the program of engineering adaptive pervasive service systems by:

– Modelling and architecting a pervasive infrastructure as a non-layered spatial substrate, hosting the execution of an ecosystem of distributed software agents, each associated to the various individual components of the infrastructure—e.g., devices, sensors, or software services.
– Exploiting the spatial substrate as a sort of shared coordination medium [11] for the agents of the ecosystem. Such a substrate embeds the basic coordination laws (*eco-laws*), which have a bio-chemical inspiration (i.e., agents manifest their activities by data-items acting as sort of chemical molecules that interact by bonding with each other and diffusing across space).
– Making the overall ecosystem behaviour be driven by the spontaneous dynamics resulting from applying the eco-laws, leading to the unplanned, i.e., self-organising, composition of distributed components, and inherently supporting dynamic context-aware and self-adaptive behaviour.

The SAPERE approach makes it easy to develop adaptive pervasive, due to both its rather intuitive programming model and its clean accompanying software engineering methodology.

Accordingly, the remainder of this paper is organised as follows. Section 2 motivates the SAPERE approach and sketches its overall agent-based architecture. Section 3 overviews and exemplifies the underlying programming model along with its coordination model based on eco-laws. Section 4 presents the methodology defined to support the design and development of complex pervasive service systems as multi-agent systems (MAS) based on the SAPERE approach. Section 5 discusses some related work, then Section 6 concludes the paper.

2 MAS for Pervasive Service Ecosystems in SAPERE

SAPERE targets emerging pervasive computing scenarios based on agent-based abstractions. This calls for specific requirements for SAPERE systems (Subsection 2.1), and also leads to a specific agent-oriented meta-model (Subsection 2.2).

2.1 Basic Requirements

The first key requirement is *situatedness* in the physical and social environment. In SAPERE pervasive systems, each agent represents individuals, software, and data tightly linked to a given space-time situation, which should affect the overall system only based on some notion of locality that can take into account physical issues (such as the position in an articulated environment) or social ones (such as who triggered some activity, and which are his/her social profile and relationships). Accordingly, the underlying meta-model should make sure that agents can access to (and influence) only a limited portion of the overall environment.

The second key requirement is *self-adaptivity.* The overall MAS should exhibit the inner ability to intercept relevant distributed situations, even those not explicitly considered at design-time, and accordingly react with no global supervision to achieve the overall system goals—both implicit and explicit ones. This should be achieved by spontaneous re-distribution and re-shaping of the overall system information and activities.

Finally, since emergent pervasive computing scenarios are based on the opportunistic encounter of devices, humans, data, and activities, with no prior knowledge of each other, a high degree of *openness* is required, which should reflect in the use of semantic-based and fully-decoupled interaction mechanisms.

2.2 The SAPERE Meta-model

Once the main requirements for SAPERE systems are introduced, the main abstractions of the SAPERE meta-model can be defined, which tailors multi-agent systems (MAS) for pervasive computing scenarios.

Agents — *Agents* are the main abstraction in the SAPERE model. As the *loci* encapsulating autonomy and control, agents are the natural means to model sensors and actuators of pervasive computing system, as well as software services (i.e., web services, situation recognisers, local monitors), and the software managing handheld devices carried by humans.

LSA — Because of the need of coordinating different kinds of entities in an open way and without global supervision, a cornerstone of the SAPERE approach is that agents manifest their existence in the MAS by a *uniform representation* called a *Live Semantic Annotation* (LSA). An LSA exposes every information about the agent (state, interface, goal, knowledge) that is pertinent for the system: it is *live* since it should continuously reflect changes in the agent state; it is *semantic* since it should be implicitly or explicitly connected to the context in which such information is produced, interpreted and manipulated; and it has the form of an *annotation*, i.e., a structured piece of information resembling a resource description—as in RDF.

LSA-space — Manifestation of LSAs is supported by the so-called *LSA-space*, acting as the true *fabric* of all interactions. There, LSAs are injected by agents, float, and evolve, ultimately reifying all the required information about system activities and processes. The LSA-space is distributed among all devices of the pervasive computing system: the portion of the LSA-space that represents a single locality of the environment is called *local LSA-space.*

LSA bonding — In order to make any agent act in a meaningful way with respect to the context in which it is situated, special mechanisms are needed to control the sphere of influence of each agent. To this end, LSAs can include *bonds* (i.e., references) to other LSAs in the same context. Only via a bond to another LSA an agent can read its information, inspect the state/interface of another agent, and act accordingly.

Eco-laws — Because of adaptivity, while agents enact their *individual* behaviour by observing their context and updating their LSAs, *global* behaviour (i.e., global coordination in the MAS) is enacted by rules manipulating the LSA-space, called *eco-laws*. Eco-laws can perform deletion/update/movement/re-bonding actions applied to a small set of LSAs in the same locality—similarly to how chemical laws affect molecules.

Thus, agents inject LSAs in the space, which by proper diffusion and aggregation eco-laws establish *fields data structures* [12,13,14] of LSAs, which cover subparts of the network and carry information about the originating LSAs (and agent) and its position in the network. Any agent interested in reading such information will then autonomously manifest this fact in its LSA, which by proper bonding eco-laws will then bond to the local LSA of the field. After all the required information has been read, the agent can affect the field originator by injecting itself an LSA, which spreads back, reach the originator's side, and is read through the same bonding mechanism.

3 Programming SAPERE Systems: API and Examples

In this section we overview how SAPERE applications can be programmed, by introducing some of the API of the SAPERE middleware and exemplifying its usage. While the whole articulation of SAPERE programming cannot be fully described here, we intend at least to give readers a clue, and also enable them to better understand the overall SAPERE development methodology.

As for any distributed environment, the execution of SAPERE applications is supported by a middleware infrastructure [15]. The infrastructure is lightweight, and enable a SAPERE node to be installed in tablets and smartphones. From the operational point of view, all SAPERE nodes are at the same level since the middleware code they run could support the same services, and provides the same set of functions—i.e., hosting the LSA space and the eco-laws engine.

From the viewpoint of the individual agents constituting the basic execution unit, middleware provides them with an API for advertising themselves via LSAs, and to support LSA continuous updating. In addition, API enables agents to detect local events, such as the change of some LSAs, or, the enactment of some eco-laws on available LSAs. Eco-laws are built as a set of rules embedded in SAPERE nodes, each hosting a local LSA-space. For each node, the same eco-laws apply to rule the dynamics of both local LSAs (in the form of bonding, aggregation, and decay), and non-locally-situated LSAs (via the spreading eco-law that can propagate LSAs through distributed nodes).

From the viewpoint of the underlying network infrastructure, the middleware transparently absorbs dynamic changes at the arrival/dismissing of supporting devices, without affecting the individual perception of the spatial environment.

3.1 The SAPERE API

In the SAPERE model, each agent executing on a node takes care of *(i)* initialising at least one LSA, and possibly more, *(ii)* injecting them on the local LSA

```
AgentNoiseSensor {
    init() {
        float nl = sample();
        injectLSA( [sensor-type = noise; accuracy = 0.1; noise-level = nl] );
    }
    run() {
        while(true) {
            sleep (100);
            float nl = sample();
            updateLSA(noise-level = nl);
}   }   }
```

Fig. 1. Pseudo-code of a noise sensor

space, and *(iii)* keeping the values of such LSAs updated to reflect its current situation. Each agent can modify only its own LSAs, and eventually read the LSAs to which has been linked to by a proper bonding eco-law. Moreover, LSAs can be manipulated by eco-laws, as explained in the following sections.

The SAPERE middleware provides agents with the following API:

– `injectLSA(lsa)` is used by agents to inject an LSA into the tuple space. Each agents must inject at least one LSA at initialisation to exist within the SAPERE ecosystem.
– `updateLSA(field, new-value)` makes agents atomically update some fields of an LSA to keep it alive. The idea is that specific threads inside agents are launched to ensure that the values of LSAs to be kept alive are promptly updated.
– A set of `onEcoLawEvent(lsa)` methods makes it possible for an agent to sense and handle whatever events occur on its LSAs. For example, the `onBond(lsa)` method allows the event represented by the LSA to be bond with another LSA matching the former.

As a first example, Figure 1 reports the (pseudo-)code of an agent that acts as a noise sensor, injecting an LSA with noise level, and periodically updating it.

3.2 Matching and Bonding

More generally, LSAs are built as descriptive tuples made by a number of fields in the form of "name-value" properties, and possibly organised in a hierarchical way: the value of a property can be a SubDescription—a set of "name-value" properties, again. By building over tuple-based models [11], the values in a LSA can be either *actual* – yet possibly dynamic and changing over time (which makes LSAs live), or *formal*, that is, not tied to any actual value unless bond to one and representing a dangling connection (typically represented with a "?").

Pattern matching between LSAs – which is at the basis of the triggering of eco-laws – happens when all the properties of a description match, i.e., when for each property whose names correspond (i.e., are semantically equivalent) the associated values match. As in classical tuple-based approaches, a formal

```
Agent AccessNoiseInformation {
    init() {
        injectLSA(sensor-type = noise; noise-level = "?");
    }
    onBond(LSA b) {
        float nl = b.noise-level();
        print("current level of noise = "+ nl);
}    }
```

Fig. 2. An agent that inject an LSA matching with that of the noise sensor and enables it to access the corresponding noise-level information

value matches with any corresponding actual value [11]. For instance, the LSA of the noise sensor in Figure 1 can match the following `(sensor-type = noise; noise-level = ?)`, expressing a request for acquiring the current noise level. The properties in the first LSA (e.g., accuracy) are not taken into account by the matching function which considers only inclusive match. The basic reaction of the LSA-space in the presence of two matching LSAs is to bond them.

Bonding upon match is the primary form of interaction among co-located agents in SAPERE—i.e., within the same LSA-space. In particular, bonding can be used to locally discover and access information, as well as to get in touch with and access local services—all of which with a single and unique adaptive mechanism. Basically, the *bonding* eco-law implements a sort of a virtual link between LSAs, whenever two LSAs (or some SubDescriptions within) match, by connecting the respective formal and actual values in a sort of bidirectional and symmetric link: the two agents holding bond LSAs can read each other's LSAs, thus enabling exchange of information.

Thus, once a formal value of an LSA matches with an actual value in an LSA it is bound to, the corresponding agent can access the actual values associated with the formal ones. For instance, the `AccessNoiseInformation` agent in Figure 2 injects an LSA matching that of Figure 1, thus enabling `AgentNoiseSensor` in Figure 1 to access the corresponding noise level information.

Bonding is automatically triggered upon match—that is, the middleware looks for possible bonding upon any relevant change to the LSAs. Analogously, debonding takes place automatically whenever matching conditions no longer hold due to some changes to the actual "live" values of some LSAs.

3.3 From Bonding to Service Composition

The above example shows how to program SAPERE agents and, depending on the LSAs injected by such agents, how bonding takes place along with exchange of information. However, it is also possible to express a formal field with the syntax "`!`", to represent a field that is formal unless the other "?" field has been bond. This makes it possible for an LSA to express parameterised services, where

"?" represents the parameter of the service, and "`!`"field represents the answer that it is able to provide once it has been filled with the parameters.

It should be noted that the bonding eco-law mechanism can be used to enable two agents to spontaneously get in touch with each other and exchange information with a single operation—and, in the case of "`!`", automatically composing two components and have the first one automatically invoking the services of the second one. That is, unlike traditional discovery of data and services, bonding makes it possible to compose services without distinguishing between the roles of the involved agents, and subsuming the traditionally-separated phases of discovery and invocation.

3.4 Aggregation, Decay, and Spreading

The additional eco-laws of *aggregation*, *spreading*, and *decay* can be triggered by agents simply by injecting LSAs with specific properties.

The *aggregation* eco-law means to aggregate LSAs together so as to compute summaries of the current system context. An agent can inject an LSA with an *aggregate* and *type* properties. The *aggregate* property identifies a function to base the aggregation upon. The *type* property identifies which LSAs to aggregate. In particular, it identifies a numerical property of LSAs to be aggregated. In the current implementation, the aggregation eco-law is capable of performing most common order and duplicate insensitive (ODI) aggregation functions [16,17].

The *decay* eco-law enables the vanishing of components from the SAPERE environment: it applies to all LSAs that specify a decay property to update the remaining time to live according to the specific decay function, or actually removing LSAs that, based on their decay property, are expired. For instance, `[sensor-type = noise; noise-level = 10; DECAY=1000]`, makes LSAs be automatically deleted after a second.

The *spreading* eco-law – unlike the two above that act on a single LSA space – enable non-local interactions, and specifically provides a mechanism to send information to remote LSA spaces, and make it possible to distribute information and results across LSA spaces. One of the primary usages of the spreading eco-law is to enable searches for components that are not available locally, and vice versa to enable the remote advertisement of services. For an LSA to be subject to the spread eco-law, it has to include a `diffusion` field, whose value (along with additional parameters) defines the specific type of propagation.

3.5 Towards Self-organisation Patterns

The eco-laws described above represent a necessary and complete to effectively support self-organising, nature-inspired interactions. In fact, by shaping LSAs so as to properly trigger eco-laws in a combined way, it is possible to realise a variety of self-adaptive and self-organising patterns.

For example, aggregation applied to the multiple copies of diffused LSAs can reduce the number of redundant LSAs so as to form a distributed *gradient* structure, also known as *computational force fields* [18]. As detailed in [19,12,13], many different classes of self-organised motion coordination schemes, self-assembly, and distributed navigation can be expressed in terms of gradients. By bringing also the decay eco-law into play, it is possible to build pheromone-based distributed data structures. Further examples can be found in [14].

4 Engineering SAPERE Systems: The Methodology

According to Osterweil [20] "software processes are software too": so, in order to build the SAPERE methodology, we follow a path that corresponds to the design of a software system. Thus, we first define the set of the SAPERE *methodology requirements* (Subsection 4.1); then we design the SAPERE *methodology process* (Subsection 4.2).

4.1 Requirements for the SAPERE Methodology

The first, obvious requirement is that the SAPERE methodology should support the design and development of SAPERE pervasive service ecosystems according to the above-mentioned meta-model (Subsection 2.2). From the analysis of the state-of-the-art in the Software Engineering area [21] the following methodology requirements can be pointed out:

– Due to the nature of the application domain, the more appealing process model is seemingly the *iterative model*, allowing engineers to iterate the different phases in order to obtain the best design.
– The SAPERE process should be organised in five main phases (*Requirements Analysis*, *Analysis*, *Architectural Design*, *Detailed Design*, and *Implementation*) in order to maintain the coherence with the general structure of standard design methodologies. This would make it easier understanding the methodology also for non-domain experts.
– The first two phases (Requirements Analysis, Analysis) should be very similar to the traditional analysis phases. On the one hand, this should make the adoption of the methodology easier to non-domain expert; on the other hand, it is generally understood that the analysis phase investigates the so called "problem domain", and the "problem" is not directly related to the technologies adopted for resolving it.
– The methodology should provide *specific activities* supporting the designer in the choice of architectural patterns and self-$*$ mechanisms, in order to address the modelling of coordination and services. Coordination should be considered as an emergent property, so that a specific self-organising pattern could be chosen in order to obtain the required coordination goal.
– Since SAPERE deals with the investigation of self-aware pervasive ecosystems, the SAPERE methodology should deal with specific activities of simulation and validation in the Architectural Design phase. In particular, simulation should take inspiration from the existing related works such as [22,23], where a suite of activities such as "Exact Verification", "Simulation", and "Tuning" are already defined in a method fragment. However, the

SAPERE methodology should not adopt the proposed fragment *as is*, but should instead provide a specific version of the aforementioned activities—namely, “Exact Prediction”, “Approximate Prediction”, “Simulation”, and “Tuning”. Also, the methodology should provide specific activities for “Validation” and “Quantitative Measures” (respectively, in the Detailed Design and in the Implementation phases) which could provide engineers with effective data and information about the behaviour of the running system.

– From the meta-model point of view, taking inspiration from the work done in the AOSE field [21], the methodology meta-model should be created according to the *transformational structure* – i.e., each phase/domain should feature *its own set of abstractions* as in Model-Driven Engineering – for the sake of clarity, and to make it easier to move from one phase to another.
– The meta-model abstractions belonging to the Requirements Analysis and Analysis phases should come both from traditional problem analysis and from some AOSE methodologies where environment abstractions and environment topology are first-class abstractions. This allows the environment to be taken into account since the first phases of the process.
– The meta-model abstractions belonging to Architectural Design and Detailed Design should be created *ex-novo* drawing from the SAPERE meta-model described in Section 2. In particular, the work done in [8] about the chemical metaphor is very useful for the identification of the design abstractions.

4.2 The SAPERE Process

The SAPERE methodology is illustrated according to the IEEE-FIPA Standard Design Process Documentation Template (DPDT) [24], developed as an internationally-recognised standard in order to facilitate the understanding of the methodology, as well as the comparison with others. For the sake of brevity, in the following we outline just the main features of the SAPERE methodology.

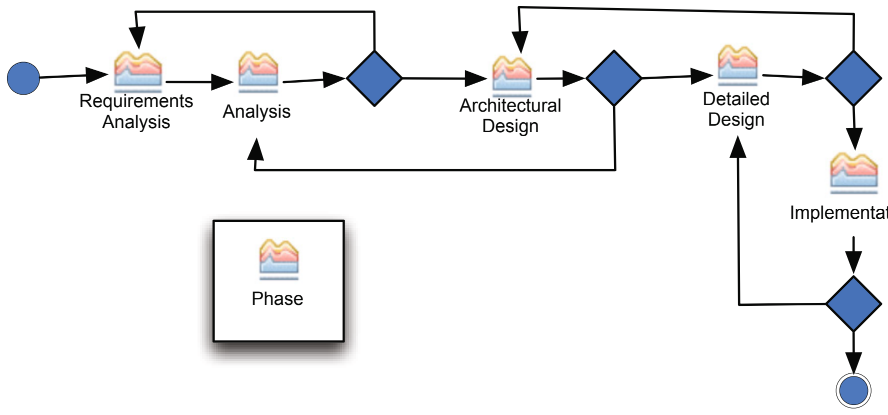

Fig. 3. The SAPERE methodology lifecycle

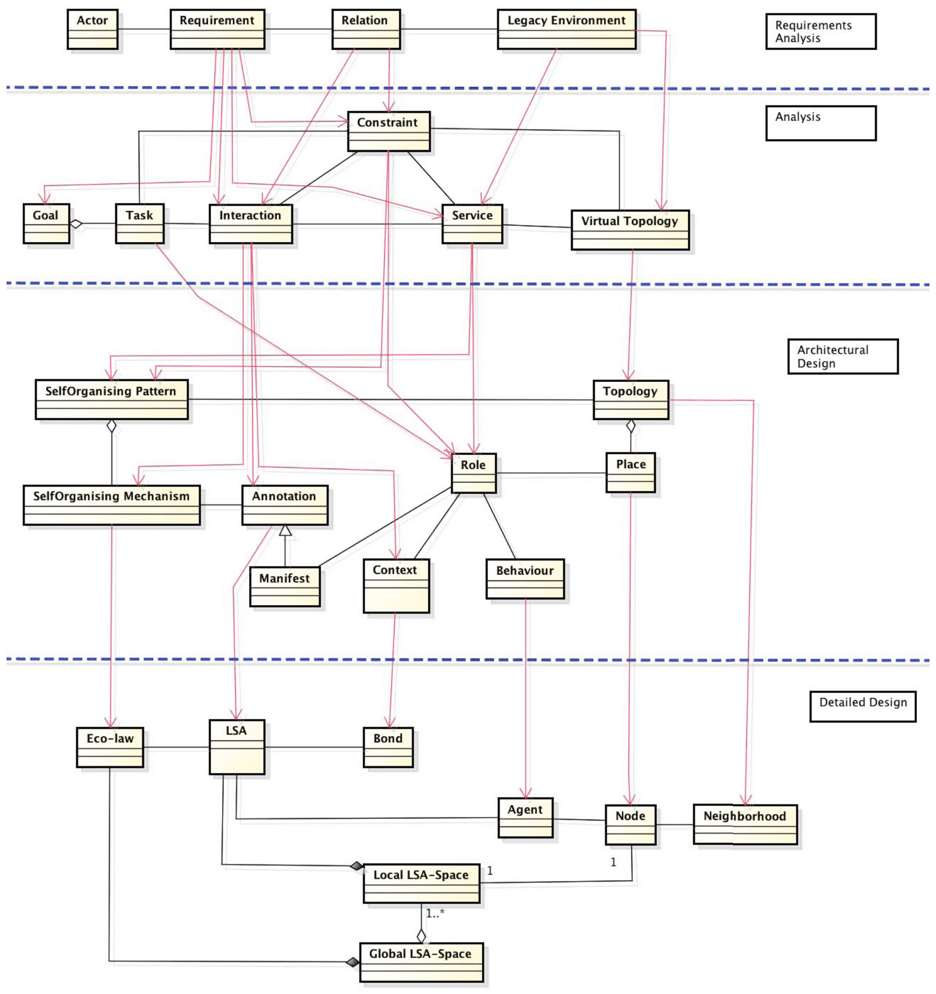

Fig. 4. The SAPERE methodology meta-model

The Lifecycle. The SAPERE methodology lifecycle is an iterative process composed by five main phases: Requirements Analysis, Analysis, Architectural Design, Detailed Design, and Implementation (Figure 3).

The Meta-model. The meta-model of the SAPERE methodology is reported in Figure 4. On the one hand, it complies with the transformational structure (see Subsection 4.1); on the other hand, it is organised in four different domains reflecting the first four methodology phases. Regarding the Implementation phase, a specific meta-model is not required here since the design abstractions have to be mapped onto the SAPERE middleware abstractions. Here we only report the ideas that inspired the meta-model construction. In particular, the abstractions of the Detailed Design phase come from the SAPERE abstract model, whereas the abstractions of the Architectural Design have many sources: *(i)* the SAPERE abstract model – *Annotation*, *Manifest*, *Context*, *Behaviour*, *Place*, *Topology* –, *(ii)* the self-organisation domain – *SelfOrganising Pattern*, and *SelfOrganising Mechanism* –, and *(iii)* the AOSE methodologies—*Role*.

For the abstractions of the Analysis phases we take inspiration from the main AOSE methodologies [21]. In particular, for the environmental and interaction aspects we adopt the SODA style, since the SODA methodology [25] specifically focuses on the modelling and design of environment and interaction [26]. Environment modelling starts since the Requirements Analysis phase (*Legacy Environment*), then during the Analysis phase we derive the services (*Service*) from both the system requirements (*Requirement*) identified in the previous phase, and from legacy resources. Also, the environment topology is modelled since the Analysis phase (*Virtual Topology*).

Interaction issues are captured in the Requirements Analysis by the *Relation* concept, which represents any kind of relationships among requirements, and between requirement and legacy environment. In the Analysis phase, the *Relation* generates – red arrow in Figure 4 – both *Interaction* and *Constraint*. *Interactions* represent the acts of interaction among *Tasks*, among *Services* and between *Tasks* and *Services*; *Constraints*, instead, enable and bound the entities' behaviour.

Finally, in order to correctly model the requirements, in the Analysis phase we decided to perform first a goal-oriented analysis (*Goal*), then to derive tasks (*Task*) by goals—as done in [27].

The Phases. Here we introduce the five SAPERE methodology phases, by shortly discussing the high-level process diagrams.

Figure 5*(left)* presents the process diagram of the Requirements Analysis phase, composed by three main activities, namely: *Requirements Modelling*, *Legacy Enviroment Modelling*, *Relations Modelling*. There, requirements, legacy resources and relations, and dependencies among them are analysed. In this phase, traditional techniques coming from the AOSE field are adopted for analysing both the requirements and the legacy environment.

Figure 5*(right)* presents the process diagram of the Analysis phase. The Analysis is composed by five main activities. In particular, *Goals Analysis* and *Task Analysis* lead the engineers to identify firstly the system's goals and then the tasks necessary to accomplish them. *Services Analysis* is devoted to derive and to analyse the system's services coming both from the legacy environment and from the system's requirements, while *Virtual Topology Analysis* analyses the system's environment topology. Finally, *Interactions Analysis* and *Constraints Analysis* respectively accounts for the interactions among system's entities and the possible constraints about entities behaviours, or about the system environment.

Figure 6 presents the process diagram of the Architectural Design phase. This phase is composed by nine main activities, namely: *Topologies Design*, *SelfOrganisations Design*, *Roles Design*, *Context Awareness Design*, *Models Extraction*, *Exact Prediction*, *Approximate Prediction*, *Simulation*, and *Tuning*. The process

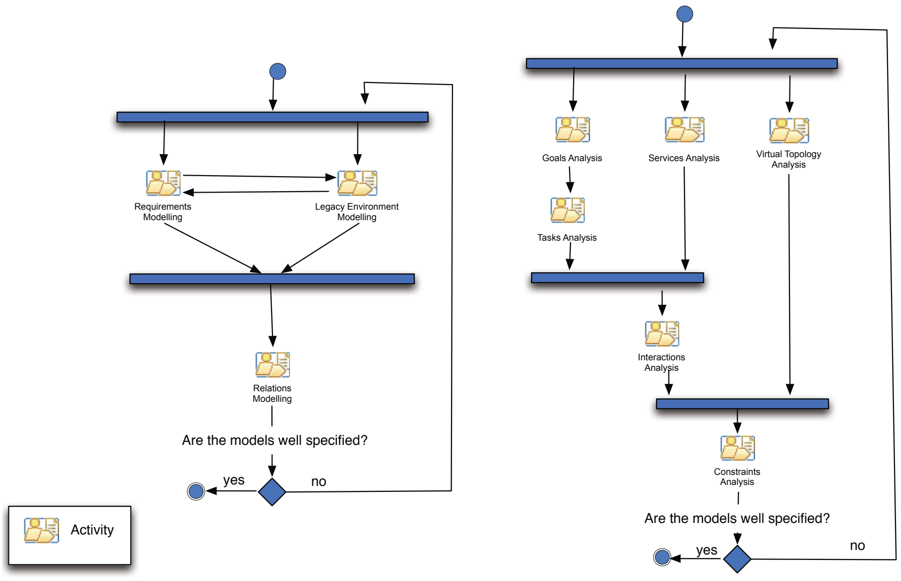

Fig. 5. Requirements Analysis *(left)* and Analysis *(right)* activities diagrams

here is more complex since the system, following problem analysis, have to be designed according to the SAPERE approach. In particular, the first four activities – *Topologies Design*, *SelfOrganisations Design*, *Roles Design*, *Context Awareness Design* – define the models for system roles (their behaviours and interactions), the self-organisation mechanisms for the services identified in the analysis, the requisite context or situation recognition in terms of roles and their communications, and the topological structure of the environment. Then, taking inspiration from [22,23], we design five activities (*Models Extraction*, *Exact Prediction*, *Approximate Prediction*, *Simulation*, and *Tuning*) devoted to system prediction and simulation. Thus, the effect of the architectural design on the system behaviour could be verified through the study of emerging properties. Adopting simulation during architectural design makes it possible for engineers the early discovery of problems due to either unsatisfactory architectural choice or inaccurate problem analysis.

Figure 7*(left)* presents the process diagram of the Detailed Design phase. This phase is composed by five main activities, namely: *Eco-Laws Design*, *Agents Design*, *Neighbourhood Design*, *Bonds Design*, *Validation*. The first four activities are devoted to the detailed design of system according to the SAPERE abstract model, while *Validation* allows engineers to effectively validate the behaviour of the whole system entities before starting the implementation phase.

Finally, Figure 7*(right)* presents the process diagram of the Implementation phase. This phase is composed by six main activities, namely: *Middleware Adaptation*, *Coding*, *WhiteBox Testing*, *BlackBox Testing*, *SystemTesting*, and *Quantitative Measures*. *Middleware Adaptation* plays a key role in this phase since in

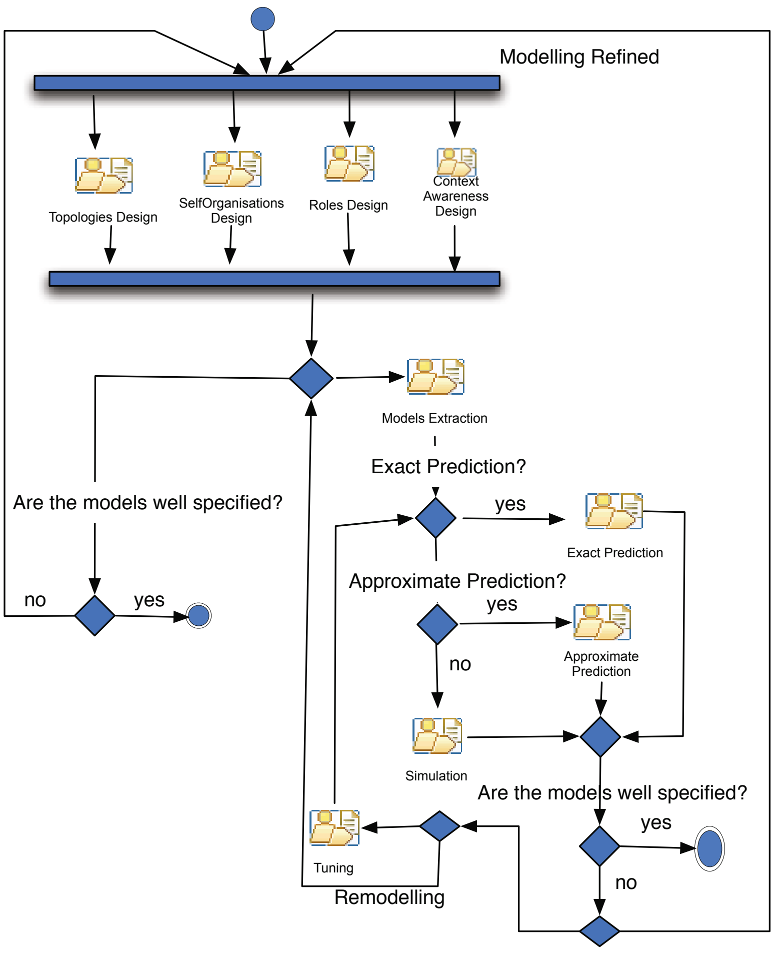

Fig. 6. The Architectural Design activities diagram

this activity the detailed design entities have to be mapped onto the middleware entities. This activity should be "trivial" – i.e., one-to-one mapping – if the middleware totally supports the detailed design entities, otherwise it could be very complex and require a lot of re-engineering work, such as the ex-novo creation of ad hoc self-organisation mechanisms. Then, *Coding* has to start before *WhiteBox Testing* and *BlackBox Testing*, but after that their executions could be interleaved. *WhiteBox Testing* represents the classical test activity conducted by the system developers during the implementation, while *BlackBox Testing* is conducted by team members not directly involved in the development of the system part under test. *SystemTesting* represents the test of the whole system for evaluating the system requirements satisfaction accuracy. Only when the system developing is concluded it is possible to execute specific Quantitative Measures – *Quantitative Measures* activity – for measuring system performances.

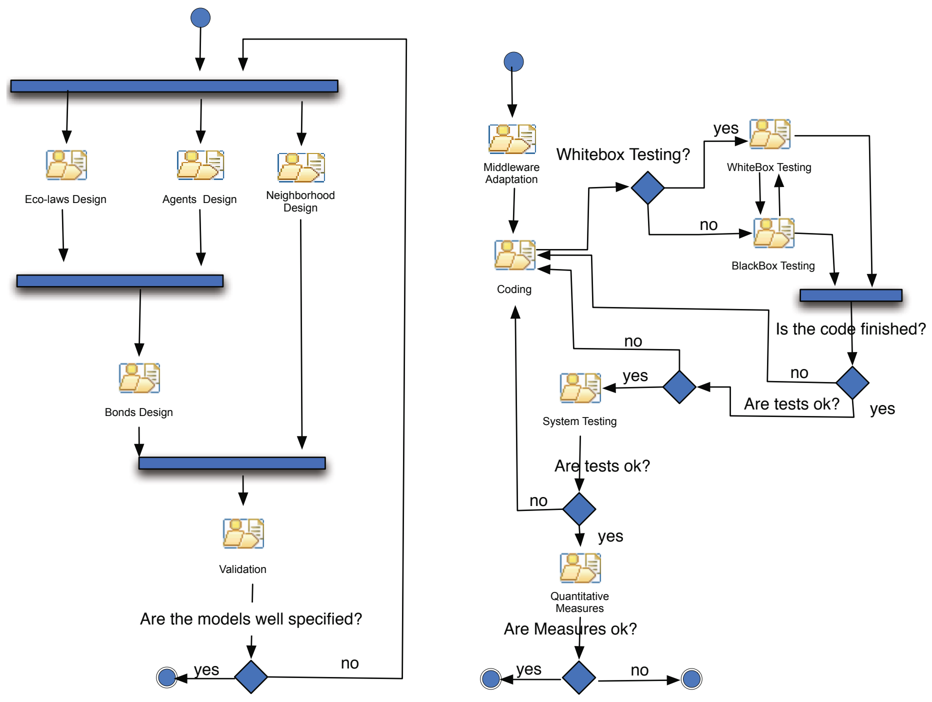

Fig. 7. The Detailed Design *(left)* and Implementation *(right)* activities diagram

5 Related Works in the AOSE Field

As far as software engineering is concerned, the key implication is that the design and development of software systems according to a (new) paradigm can by no means rely on conceptual tools and methodologies conceived for a totally-different (old) paradigm [28]. Even though it is indeed possible to develop a complex distributed system in terms of objects and client-server interactions, such a choice appears odd and complicated when the system is a Multi-Agent System (MAS), or, it can be assimilated to a MAS. Rather, a brand new set of conceptual and practical tools – specifically suited to the agent-oriented abstractions – is needed to facilitate, promote, and support the development of MASs, and to fulfil the huge potential of agent-based computing as a general-purpose approach to the modelling and engineering of complex systems.

The definition of agent-specific methodologies is definitely one of the most explored topics in Agent-Oriented Software Engineering (AOSE), and a large number of AOSE methodologies – describing how the process of building a MAS should/could be organised – has been proposed in the literature, which should be compared to the SAPERE approach presented in this paper. For a rather exhaustive survey of all the related activities in the AOSE field, we refer the interested reader to [21].

Meta-model. In the same way as the SAPERE one, AOSE methodologies typically start by defining their own meta-model, identifying the basic abstractions to be exploited in development (e.g., agents, roles, environment, organisational structures). Based on this, they exploit and organise such abstractions so as to define guidelines on how to proceed in the analysis, design, and development, and on the output to produce at each stage.

Actually, several works [29,30] are focussing on the identification of appropriate meta-models for AOSE methodologies and process models—where a meta-model is intended as a rational analysis and identification of the abstractions used in MAS development. Those efforts aims at unifying the different abstractions adopted in existing methodologies and the process models, and also at identifying which relationships may exist among them. This may be used to better understand the real usefulness of the abstractions, and also to improve or unify processes and methodologies. Furthermore, those effort may help researchers and practitioners to identify and develop conceptual instruments and practical tools for an efficient processes management.

Process model. Among the different methodologies developed both in the traditional software engineering – such as Rational Unified Process (RUP) [31], OPEN [32], Object Process Methodology (OPM) [33], OMT [34], Fusion [35] – and in the agent-oriented world – such as PASSI [36], Gaia [37], INGENIAS [38], MESSAGE [39], Adelfe [40], Tropos [41], MaSE [42], SODA [25,43,44,45] etc. –, there is a general agreement on organising the methodology process according to two main phases: Analysis and Design. However, the different methodologies often introduce other phases or sub-phases. In particular, the Analysis phase is typically split into Requirements Analaysis and Analysis, while the Design is typically organised in terms of Architectural Design and Detailed Design [46]. In addition, different methodologies guide the system development until the implementation phase—among them RUP, OPEN, PASSI, INGENIAS, and ADELFE.

As discussed in Subsection 4.2, according to the general agreement on the main phases of a development process, the SAPERE methodology is organised in five main phases: Requirements Analysis, Analysis, Architectural Design, Detailed Design, and Implementation.

Meta-model vs. process model. Quite different and heterogeneous abstractions are adopted by the different methodologies for modelling complex MAS: typically, in the AOSE world, each methodology defines its own set of abstractions. This is why AOSE methodologies typically start by providing the so-called *abstractions meta-model* [30,47] that shows all the abstractions adopted by the methodology, along with their mutual relationships.

Names for abstractions are used quite liberally: different names sometimes refer to similar abstractions, whereas identical names may denote quite diverse abstractions—even within one single methodology, when the same name is sometimes used for abstractions holding different meanings, depending on the different process phases they belong to. For instance, the "agent" concept, quite unsurprisingly, is exploited by all AOSE methodologies. However, whereas in some methodologies – such as PASSI, MaSE, and ADELFE – the agent abstraction appears since the Analysis phase, in other methodologies – such as Tropos, Gaia, and SODA – the agent is a concept occurring only in the Design phases. So, the issue is not merely which abstractions meta-model is adopted by a given AOSE methodology: but, more precisely, which abstractions are used in each phase of the methodology, and how the different resulting abstractions meta-models relate to each other.

Even more, also the structure of the abstractions meta-model differs a lot among the methodologies. For example, PASSI and SODA adopt a "transformational" structure – i.e., each phase/domain has its own particular set of abstractions – taking inspiration from the Model-Driven Engineering ideas. Instead, other methodologies such as ADELFE use the same set of abstractions, which are refined by each phase. For a more detailed work about the study, comparison, and fusion of some AOSE methodologies meta-models, we refer the interested reader to [27].

Taking inspiration from the work done in the AOSE field, the SAPERE methodology meta-model was in fact defined according to the transformational structure: this allow each phase to be more clearly specified, and makes it easier to move from one phase to another—see Subsection 2.2.

6 Conclusion

The definition of a novel and coherent methodological process for the engineering of SAPERE pervasive service ecosystems was the main motivation behind this work. In order to allow the reader to fully understand the SAPERE process, in this paper we first introduce the SAPERE model, then discuss how a SAPERE system could be programmed, by providing some simple examples, finally we illustrate the SAPERE methodology, by defining the software development process according to the IEEE-FIPA Standard Design Process Documentation Template (DPDT) [24].

The space available for this paper is obviously not enough to provide the reader with all the details of the SAPERE methodology: for a full account of the SAPERE methodology we refer the interested reader to [48]. In this paper we discuss the main issues of the engineering of pervasive service ecosystems according to the SAPERE approach, thus showing how agent-oriented technologies and methodologies can be effective in the design and development of complex software systems.

Acknowledgements. This work has been supported by the EU-FP7-FET Proactive project SAPERE – Self-Aware PERvasive service Ecosystems, under contract no. 256873.

References

1. Krumm, J.: Ubiquitous advertising: The killer application for the 21st century. IEEE Pervasive Computing 10(1), 66–73 (2011)
2. Zambonelli, F.: Toward sociotechnical urban superorganisms. Computer 47(8), 76–78 (2012)
3. Zambonelli, F.: Pervasive urban crowdsourcing: Visions and challenges. In: 2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp. 578–583. IEEE CS Press (2011)
4. Zambonelli, F., Viroli, M.: A survey on nature-inspired metaphors for pervasive service ecosystems. International Journal of Pervasive Computing and Communications 7(3), 186–204 (2011)
5. Babaoglu, O., et al.: Design patterns from biology for distributed computing. ACM Transaction on Autonomous Adaptive Systems 1(1), 26–66 (2006)
6. Mamei, M., Menezes, R., Tolksdorf, R., Zambonelli, F.: Case studies for self-organization in computer science. Journal of Systems Architecture 52(8), 443–460 (2006)
7. Kari, L., Rozenberg, G.: The many facets of natural computing. Communications of the ACM 51, 72–83 (2008)
8. Zambonelli, F., Castelli, G., Ferrari, L., Mamei, M., Rosi, A., Di Marzo Serugendo, G., Risoldi, M., Tchao, A.E., Dobson, S., Stevenson, G., Ye, Y., Nardini, E., Omicini, A., Montagna, S., Viroli, M., Ferscha, A., Maschek, S., Wally, B.: Self-aware pervasive service ecosystems. Procedia Computer Science 7, 197–199 (2011), Proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 2011)
9. Parunak, V.: Go to the ant: Engineering principles from natural multi-agent systems. Annals of Operations Research 75, 69–101 (1997)
10. Omicini, A.: Nature-inspired coordination for complex distributed systems. In: Fortino, G., Badica, C., Malgeri, M., Unland, R. (eds.) Intelligent Distributed Computing VI. SCI, vol. 446, pp. 1–6. Springer, Heidelberg (2012)
11. Gelernter, D.: Generative communication in Linda. ACM Transactions on Programming Languages and Systems 7(1), 80–112 (1985)
12. Mamei, M., Zambonelli, F.: Programming pervasive and mobile computing applications: The TOTA approach. ACM Transactions on Software Engineering and Methodology 18(4) (July 2009)
13. Viroli, M., Casadei, M., Montagna, S., Zambonelli, F.: Spatial coordination of pervasive services through chemical-inspired tuple spaces. ACM Transactions on Autonomous and Adaptive Systems 6(2), 14:1–14:24 (June 2011)
14. Fernandez-Marquez, J.L., Di Marzo Serugendo, G., Montagna, S., Viroli, M., Arcos, J.L.: Description and composition of bio-inspired design patterns: A complete overview. Natural Computing 12(1), 43–67 (2013)
15. Zambonelli, F., Castelli, G., Mamei, M., Rosi, A.: Integrating pervasive middleware with social networks in sapere. In: 2011 International Conference on Selected Topics in Mobile and Wireless Networking, pp. 145–150 (October 2011)
16. Nath, S., Gibbons, P.B., Seshan, S., Anderson, Z.R.: Synopsis diffusion for robust aggregation in sensor networks. In: 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), pp. 250–262. ACM, New York (2004)
17. Bicocchi, N., Mamei, M., Zambonelli, F.: Self-organizing virtual macro sensors. ACM Transaction on Autonomous Adaptive Systems 7(1) (2012)

18. Mamei, M., Zambonelli, F.: Field-Based Coordination for Pervasive Multiagent Systems. In: Models, Technologies, and Applications. Springer Series in Agent Technology. Springer (March 2006)
19. Beal, J., Bachrach, J.: Infrastructure for engineered emergence on sensor/actuator networks. IEEE Intelligent Systems 21(2), 10–19 (2006)
20. Osterweil, L.J.: Software processes are software too. In: 9th International Conference on Software Engineering (ICSE 1987), pp. 2–13. IEEE Computer Society Press, Los Alamitos (1987)
21. Molesini, A., Omicini, A.: Early methodology. Technical Report TR.WP1.2012.6, EU-FP7-FET Proactive project SAPERE Self-Aware PERvasive service Ecosystems (2012), http://www.sapere-project.eu/TR.WP1.2012.6.pdf
22. Gardelli, L., Viroli, M., Casadei, M., Omicini, A.: Designing self-organising environments with agents and artefacts: A simulation-driven approach. International Journal of Agent-Oriented Software Engineering 2(2), 171–195 (2008), Special Issue on Multi-Agent Systems and Simulation
23. Molesini, A., Casadei, M., Omicini, A., Viroli, M.: Simulation in agent-oriented software engineering: The SODA case study. Science of Computer Programming (August 2011), Special Issue on Agent-oriented Design methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments
24. IEEE-FIPA: Design Process Documentation Template (January 2012), http://fipa.org/specs/fipa00097/SC00097B.pdf
25. SODA: Home page, http://soda.apice.unibo.it
26. Molesini, A., Omicini, A., Viroli, M.: Environment in Agent-Oriented Software Engineering methodologies. Multiagent and Grid Systems 5(1), 37–57 (2009), Special Issue "Engineering Environments in Multi-Agent Systems
27. Dalpiaz, F., Molesini, A., Puviani, M., Seidita, V.: Towards filling the gap between AOSE methodologies and infrastructures: Requirements and meta-model. In: Baldoni, M., Cossentino, M., De Paoli, F., Seidita, V. (eds.) 9th Workshop From Objects to Agents (WOA 2008), Palermo, Italy, Seneca Edizioni, pp. 115–121 (November 2008)
28. Zambonelli, F., Omicini, A.: Challenges and research directions in agent-oriented software engineering. Autonomous Agents and Multi-Agent Systems 9(3), 253–283 (2004), Special Issue: Challenges for Agent-Based Computing
29. Henderson-Sellers, B.: Evaluating the feasibility of method engineering for the creation of agent-oriented methodologies. In: Pěchouček, M., Petta, P., Varga, L.Z. (eds.) CEEMAS 2005. LNCS (LNAI), vol. 3690, pp. 142–152. Springer, Heidelberg (2005)
30. Bernon, C., Cossentino, M., Gleizes, M.P., Turci, P., Zambonelli, F.: A study of some multi-agent meta-models. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382, pp. 62–77. Springer, Heidelberg (2005)
31. Kruchten, P.: The Rational Unified Process: An Introduction, 3rd edn. Addison-Wesley Professional (December 2003)
32. OPEN: Home page, http://www.open.org.au/
33. Dori, D.: Object-Process Methodology: A Holistic System Paradigm. Springer (2002)
34. Rumbaugh, J.E., Blaha, M.R., Premerlani, W.J., Eddy, F., Lorensen, W.E.: Object-Oriented Modeling and Design. Prentice-Hall (1991)
35. Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., Jeremaes, P.: Object-Oriented Development. The Fusion Method. Prentice-Hall (1994)
36. Cossentino, M.: From requirements to code with the PASSI methodology. In: [49], ch. IV, pp. 79–106

37. Zambonelli, F., Jennings, N., Wooldridge, M.: Multiagent systems as computational organizations: the Gaia methodology. In: [49], ch. VI, pp. 136–171
38. Pavòn, J., Gòmez-Sanz, J.J., Fuentes, R.: The INGENIAS methodology and tools. In: [49], ch. IX, pp. 236–276
39. Garijo, F.J., Gòmez-Sanz, J.J., Massonet, P.: The MESSAGE methodology for agent-oriented analysis and design. In: [49], ch. VIII, pp. 203–235
40. Picard, G., Bernon, C., Gleizes, M.P.: Cooperative agent model within ADELFE framework: An application to a timetabling problem. In: Jennings, N.R., Sierra, C., Sonenberg, L., Tambe, M. (eds.) AAMAS,, July 19-23, vol. 3, pp. 1506–1507. ACM Press, New York (2004)
41. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: Tropos: An agent-oriented software development methodology. Autonomous Agent and Multi-Agent Systems 8(3), 203–236 (2004)
42. Wood, M.F., DeLoach, S.A.: An overview of the multiagent systems engineering methodology. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957, pp. 207–221. Springer, Heidelberg (2001)
43. Omicini, A.: SODA: Societies and infrastructures in the analysis and design of agent-based systems. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957, pp. 185–193. Springer, Heidelberg (2001)
44. Molesini, A., Omicini, A., Ricci, A., Denti, E.: Zooming multi-agent systems. In: Müller, J.P., Zambonelli, F. (eds.) AOSE 2005. LNCS, vol. 3950, pp. 81–93. Springer, Heidelberg (2006)
45. Molesini, A., Omicini, A., Denti, E., Ricci, A.: SODA: A roadmap to artefacts. In: Dikenelli, O., Gleizes, M.-P., Ricci, A. (eds.) ESAW 2005. LNCS (LNAI), vol. 3963, pp. 49–62. Springer, Heidelberg (2006)
46. Cernuzzi, L., Cossentino, M., Zambonelli, F.: Process models for agent-based development. Engineering Applications of Artificial Intelligence 18(2), 205–222 (2005)
47. Cossentino, M., Gaglio, S., Galland, S., Gaud, N., Hilaire, V., Koukam, A., Seidita, V.: A MAS metamodel-driven approach to process fragments selection. In: Luck, M., Gomez-Sanz, J.J. (eds.) AOSE 2008. LNCS, vol. 5386, pp. 86–100. Springer, Heidelberg (2009)
48. Molesini, A., Omicini, A., Viroli, M., Pianini, D., Montagna, S.: The complete methodology. Technical Report TR.WP1.2013.1, EU-FP7-FET Proactive project. SAPERE Self-Aware PERvasive service Ecosystems (2013), `http://www.sapere-project.eu/TR.WP1.2013.1.pdf`
49. Henderson-Sellers, B., Giorgini, P. (eds.): Agent Oriented Methodologies. Idea Group Publishing, Hershey (2005)