Self-Managing and Self-Organising Mobile Computing Applications: a Separation of Concerns Approach

Jose Luis Fernandez-Marquez  
University of Geneve  
1227 Route de Drize, 7  
Carouge, Switzerland  
joseluis.fernandez@unige.ch

Giovanna Di Marzo Serugendo  
University of Geneve  
1227 Route de Drize, 7  
Carouge, Switzerland  
Giovanna.DiMarzo@unige.ch

Graeme Stevenson  
School of Computer Science  
University of St Andrews  
United Kingdom  
gaime.stevenson@st-andrews.ac.uk

Juan Ye  
School of Computer Science  
University of St Andrews  
United Kingdom  
juan.ye@st-andrews.ac.uk

Simon Dobson  
School of Computer Science  
University of St Andrews  
United Kingdom  
simon.dobson@st-andrews.ac.uk

Franco Zambonelli  
University of Modena and Reggio Emilia  
42122 Via Amendola, 2  
Reggio Emilia, Italy  
franco.zambonelli@unimore.it

ABSTRACT

Although the research area of self-organising systems is well established, their construction is often ad hoc. Consequently, such software is difficult to reuse across applications that require similar functionality of have similar goals. The development of self-organising applications and, a fortiori, self-organising mobile applications is therefore limited to developers who are experts in specific self-organising mechanisms. As a first step towards addressing this, this paper discusses the notion of self-organising mechanisms provided as services for building higher level functionality in a modular way. This paper suggests separation of concerns. Additionally, because of the dynamic and heterogeneous nature of mobile networks, services need to adapt themselves in order to ensure both functional and non-functional requirements. This paper discusses whether the self-management of self-organising mobile applications can be achieved in a modular fashion, via the self-management of low level self-organising services it employs, rather than considering the management of the complex system as a whole. We empirically investigate two non-functional aspects: resource optimisation and accuracy.

1. INTRODUCTION

Devices with wireless communication capability and computational power such as mobile phones, tablets, and recently, cars are gaining the capability to build large, dense, opportunistic infrastructures, such as Mobile Ad-Hoc Networks (MANETS) and Vehicular Ad-Hoc Networks (VANETS), over which a wide range of novel applications will be deployable.

Amongst the challenges facing these infrastructures is the need to support application requirements such as robustness, scalability, and adaptability in the face of dynamic behaviour: node mobility and churn, variable node densities, and host heterogeneity.

Self-organising systems (SOS) [37] provide a promising approach to architecting services for such infrastructures. Self-organising systems consist of autonomous entities – software agents, in this case physically distributed – that collaborate to provide emergent and collaborative behaviour without global state or control. Self-organisation principles can be applied at several levels, from self-organising design patterns that describe re-usable solutions for recurrent problems in engineering self-organising systems [13, 12, 15], to execution models that provide new paradigms for computing self-organising applications [33, 9, 14], and self-organising methodologies [31].

These bottom-up techniques are based on local interactions between neighbours, which contribute to achieving both desired functionality and non-functional properties such as robustness, scalability, and adaptability. However these are not automatic outcomes of their application and although obtained as a result of the self-organisation process are limited in their scope [6]. Many self-organisation techniques must be optimised for their operating environment and are not intended to overcome all possible environmental constraints. Additionally, self-organising systems are often developed in ad-hoc way, producing software that is difficult to maintain and reuse across applications.

We identify three key issues:

- There is a lack of separation of concern and reuse of functionality in engineering self-organising systems.
- Self-organising algorithms are very sensitive to their parameters, which need to be tuned both at design time and run-time based on perceived context.
It is generally the case that no single implementation of a basic self-organising mechanism, even when finely tuned, is capable of handling all possible operating environments, thus it may be desirable to switch between different implementations based on context perceived from an agent’s operating environment. This allows the targeting of accuracy, performance optimisation, availability or accessibility of a service.

How to engineer self-organising mobile applications in a modular way, using operators as building blocks for higher level functions is still an open challenge. Additionally, if we think about building blocks acting as services for composing higher level services or application, these services should ensure not only functional requirements, but also non-functional ones, by adapting their behaviour depending on contextual information (e.g. network mobility, speed of nodes, or density of nodes).

Such self-organising services require self-management in order to flexibly handle dynamic environments with little human intervention and support application reuse.

This paper investigates whether the self-management of complex SOS can be achieved in a modular fashion: through the use of reusable self-organising services and through the self-management of the self-organising services. Thus, we implement policies for adapting low level mechanisms depending on contextual information (e.g. information availability, or density of nodes) improving non-functional aspects (e.g. resource optimisation and accuracy), and we analyse the effect on higher level mechanism or services.

The paper is structured as follows. Section 2 discusses related works. Section 3 highlights separation of concerns, and discusses the notions of self-organising mechanisms as services, non-functional aspects and self-management as a responsibility of the underlying environment. Section 4 discusses implementation aspects, then Section 5 presents a proof of concept mobile application and experimental results, highlighting three levels of self-organising services, each using the level below: from spreading, to dynamic gradient to chemotaxis.

Figure 1: Self-organising Design Patterns [15]

- It is generally the case that no single implementation of a basic self-organising mechanism, even when finely tuned, is capable of handling all possible operating environments, thus it may be desirable to switch between different implementations based on context perceived from an agent’s operating environment. This allows the targeting of accuracy, performance optimisation, availability or accessibility of a service.

2. RELATED WORK

This section briefly reviews the state of the art related to the modular engineering of SOS, as well as current approaches to non-functional aspects such as optimisation and adaptation to dynamic conditions.

2.1 Separation of concerns in SOS

Separation of concerns in engineering self-organising systems has not been widely addressed in the literature. Fernandez-Marquez et al. [14] propose the use of low level self-organising mechanisms to build higher-level self-organising mechanisms and functionalities. However, this work does not discuss how these mechanisms should be implemented in order to deal with different scenarios, and how they can be reused by applications under different environmental conditions. Pursuing this idea, Giovanna Di Marzo Serugendo et al. [7] present the notion of self-organising services—proposing self-organising mechanisms as services that can be reused across many applications. This work does not discuss self-managing and non-functional issues. In this paper, we leverage from these works: we focus on the implementation of these self-organising services and on their self-management and adaptation to different environmental conditions, such as node speed, densities, or availability of contextual information.

2.2 Optimisation and adaptation in SOS

We start with the one of the first and most relevant self-organising system, Ant Colony Optimisation (ACO), which was proposed in 1999 to solve shortest path problems on a graph for instances of the Travelling Salesman Problem (TSP) [8]. It demonstrates that each instance of the TSP requires ACO parameters to be tuned in order to achieve efficiency. Many extensions of ACO algorithms such as [17] and [38] have proposed different approaches to adapt the system to different problems without requiring human intervention to reset the parameters and perform well.

A number of self-organising mechanisms have been proposed since ACO including Particle Swarm Optimisation (PSO) [19], Chemotaxis [25], Flocking [27], and Morphogenesis [21]. Most of these are similarly challenged to handle different scenarios, with optimal parameterisation dependent on the specific problem being targeted. Many extensions of the algorithms have been proposed in order to mitigate this concern, resulting in a large number of implementations that are difficult to compare because of their context dependent behaviours. For example implementation, see [3, 10, 16].

In this paper, analogously to the above related work, we target the adaptation of self-organising systems in order to improve non-functional aspects, such as their performance and behaviour adaptation in different scenarios. Instead of extending specific high-level self-organising mechanisms, we first build such high-level mechanisms in a modular fashion and in the form of services (e.g., chemotaxis is built from using a gradient and spreading service). Next, we tackle the self-management issues of performance and adaptation by providing self-management at each level: from spreading to gradient to chemotaxis itself.

3. SEPARATION OF CONCERNS

This section expands our main points for addressing separation of concerns in mobile applications: (1) self-organising mechanisms provided as services (Section 3.1); (2) separating functional and non-functional aspects (Section 3.2); (3) delegating self-management of low-level services to the underlying environment (Section 3.3); and finally (4) developing a layered self-management framework where self-managing policies apply at each layer (Section 3.4).

3.1 Self-organising mechanisms as services

Fernandez et al. [15] identified a catalogue of self-organising mechanisms and described the relations between...
them, as illustrated in Figure 1. This catalogue shows the boundaries between mechanisms and how basic mechanisms are used to compose more complex ones. Leveraging this observation, self-organising mechanisms can clearly be delineated and distinguished from each other, paving the way towards modular designs and separation of concerns in the engineering of self-organising complex applications. Contrastingly, we could, for instance, implement an Ant Colony algorithm where the responsibility of Spreading, Aggregating and Evaporating pheromones are embedded in the agents (ants). However, that prevents reuse of code and modular design of the application.

Following this idea, a middleware for engineering self-organising systems [39] has been developed in which, the basic mechanisms are provided as “core” services. Higher-level self-organising mechanisms and applications rely on these low-level mechanisms, and use and activate them as operators in order to build modular self-organising systems. Here, the responsibility of the middleware is to provide a set of low level services (e.g. spreading, aggregation, or evaporation) that higher level services and application can reuse to design and implement self-organising behaviours.

To illustrate these ideas in more depth, we now consider how three patterns – Spreading, Gradient and Chemotaxis – are linked to each other and how they can be designed and provided as modular services. The choice of these mechanisms for mobile computing applications is motivated by previous work in mobile, pervasive and context-aware scenarios [24], where spatial structures [2, 1] spread among mobile devices and maintain themselves despite the mobility or changing density of the nodes.

**Spreading.** A spreading mechanism, also called flooding or propagation is a mechanism where a piece of information is periodically sent by nodes, eventually reaching all the nodes in the infrastructure.

Spreading algorithms have been widely used as a basis for implementing routing protocols in mobile networks, such as, DSR [18] and AODV [29]. There are many different schemes for implementing propagation: probabilistic, counter based, distance based or position based propagation [26, 36]. Each scheme has the goal of reducing the number of messages transmitted and thus, the chance of collision and contention.

Spreading is a key pattern in self-organising systems because almost all self-organising algorithms use it to implement inter-entity communication. It is provided as a service or as an operator ready to be used by the underlying middleware, and has the responsibility of propagating or diffusing data for higher-level services or applications.

**Gradient.** The Gradient Pattern [15] is an extension of the Spreading Pattern where propagated information provides additional information about the sender’s distance: either a distance attribute is added to the information; or the value of the information is modified such that it reflects its concentration - higher concentration values meaning the sender is closer, such as in ant pheromones.

The Gradient Pattern has been used in problems such as coordination of swarms of robots [28], coordination of agents in video games [23], or routing in ad hoc networks [30].

In terms of services, the Gradient service uses the Spreading service to diffuse itself in a distributed environment.

**Chemotaxis.** The Chemotaxis Pattern, proposed by Nagpal [25], provides a mechanism to perform motion coordination in large scale systems. The Chemotaxis Pattern extends the Gradient Pattern: agents use the gradient direction to decide their direction of movement.

The concentration of a gradient guides the agents’ movements in three different ways, as shown in Figure 2: (1) attractive movement, where agents change their positions by following higher gradient values, (2) repulsive movement, where agents follow lower gradient values, and (3) equipotential movement, when agents follow gradients between thresholds.

Chemotaxis has been used by Mamei et al. [22] to coordinate the position of a swarm of simple mobile robots, Violi et al. [34], where chemotaxis is applied to route messages in pervasive computing scenarios, and Fernandez-Marquez et al. [11] to find diffuse event sources in noisy wireless sensor networks.

The Chemotaxis service uses the lower-level Gradient service (triggered previously by other parts of the system or the application), which in turn uses the Spreading for its own propagation.

### 3.2 Non-functional aspects and tradeoffs

Self-organising systems have been shown to adapt in response to environmental changes, be scalable and fault-tolerant. However, each self-organising algorithm faces those problems partially and not for all possible environmental conditions. Many proposals have been presented to improve the performance of self-organising systems by tuning the algorithms, allowing the algorithms to converge faster or to deal with a wider number of problems, de facto improving the adaptiveness of the algorithms.

Services that provide functionality achieved through self-organising algorithms often must ensure a certain quality of service when dealing with dynamic environments, and thus satisfy both functional and non-functional requirements.

For example, in the case of Spreading, a service could employ a probabilistic propagation scheme fixed with a low propagation probability in a very dense network, ensuring that all nodes receive the information. However, the same probability in a less dense network can reduce dramatically the reachability of the spread, preventing many nodes from receiving the information. Alternatively we could use a location-based spreading algorithm to improve performance, but if GPS information is unavailable at some nodes, the algorithm’s performance will decrease. Thinking modularly, a Spreading service is responsible for ensuring that all connected nodes receive the information. It must control its own behaviour, i.e. the probability of propagation, or the choice of algorithm in order to deliver the correct
functionality and efficiently use resources when dealing with dynamic and heterogeneous environments.

A self-organising service should contain self-management properties in order to adapt parameters when the environmental conditions require it, and to switch among algorithms depending on contextual information available.

As another example, we consider a Dynamic Gradient that maintains and updates a spatial structure that provides an estimation about the direction and distance from the gradient source. It has been demonstrated that dynamic gradients are able to deal with network mobility, however, its parameters play a key role. If a dynamic gradient is updated at a high frequency its bandwidth usage is high, however if a dynamic gradient is updated at a low frequency and the network is highly dynamic, the gradients paths will frequently not reflect the true state of the world, potentially affecting the quality of service available to applications or higher-level services that use it. When a gradient is provided as a service, the application requesting the service must not be in charge of setting the update frequency, neither should it decide on the best gradient implementation. A gradient provided as a service must be able to control its own behaviour in order that applications may rely on it.

The trade-offs between parameter values is also highlighted in the design pattern catalogue mentioned above [15], under the Forces field of the pattern description. Forces refers to the important parameters or trade-offs that must be taken into account during the implementation (usually contextual information). Forces are extremely important in order to ensure functional and non-functional aspects of each pattern. Although the forces can be established at application design time, in the experimental section we demonstrate that forces need to be adaptive depending on contextual information. This is where self-management plays a key role—controlling the implementation selection and tuning its parameters in order to increase the performance of the system to satisfy service, application, and user requirements.

Additional information about other non-functional aspect, such as, privacy can be found in [40, 4].

### 3.3 An active computational environment

Weyns et al. review the responsibilities of the environment in multi-agent systems and define the computational environment as: “...a first-class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources” [35]. They emphasise that all responsibilities not managed by the environment (i.e. by the middleware infrastructure) need to be addressed by the agents. Thus, the computational environment plays a key role as an active agent in the engineering of multi-agent systems.

As we have seen above, in addition to the functional aspects of these services, it is also important to consider non-functional requirements, such as bandwidth usage, scalability, resource consumption, and fault tolerance. This is why, in the same way that low-level self-organising mechanisms are provided as services by the middleware, the middleware must also be in charge of ensuring non-functional requirements of these services and optimising resource use to guarantee the scalability of applications running on it.

### 3.4 Layered Self-Management

![Figure 3: Self-management at different levels](image)

We identify and analyse two different ways of managing the adaptation of low-level self-organising services to improve the non-functional aspects of our layered self-management framework for which the middleware is responsible: (1) parameter adaptation - almost all self-organising algorithms need to tune their parameters depending on the environmental conditions. Here, we propose that the middleware, as a first class entity, is in charge of managing the adaptation of parameters in order to optimise the performance of the services it runs and guarantee the harmony between different applications running on the middleware; (2) service switching - different algorithms can satisfy similar functional requirements (e.g., spreading can be configured to be location based or probabilistic), however the performance of non-functional requirements may be different.

The solution presented in this paper can be extended to all patterns presented in Figure 1. Here, we focus on one design pattern at each level: Spreading, Gradient, and Chemotaxis, as shown in Figure 3.

Services proposed at different levels come with different implementations. The computational environment automatically switches between them in order to satisfy functional and non functional requirements. As we described in the previous section, Spreading can be implemented by different, context-aware algorithms, and thus provided by different instances; each being the best choice for a specific scenario. Based on contextual information, the middleware switches among the different instances of the Spreading service depending on the situation. Additionally, each instance of the service must be able to adapt its own parameters. For example, a probabilistic propagation must be able to adapt the probability in order to reduce the number of messages sent and to keep a high reachability.

A Gradient service must be able to switch between static and dynamic gradients depending on the perceived mobility of the network. Moreover, the dynamic gradient implementation should be able to adapt the frequency of gradient updates accordingly with the speed of the nodes.

### 4. IMPLEMENTATION

The above proposal for self-managing and self-organising
services has been implemented as an extension of the SAPERE\textsuperscript{1} middleware [39].

The SAPERE middleware runs on two platforms: (1) A Java middleware for laptops and mobile Android devices (i.e. tablets and mobile phones), and (2) in an opportunistic network environment simulator called The ONE [20] where each simulated mobile device executes an instance of the actual SAPERE middleware.

This extension of The ONE with the SAPERE middleware, so called, The ONE-SAPERE\textsuperscript{2} plays a key role in prototyping and validating new functionalities before they are released for use in real mobile devices.

4.1 The SAPERE middleware

The SAPERE middleware provides an active computational environment where information injected by agents is subject to chemical reactions. The active computational environment, implemented as a tuple space, is in charge of executing environmental tasks, such as Spreading, Evaporation, and Aggregation of information (i.e. low-level self-organising mechanisms provided as core services by the middleware). These services are implemented as chemical reactions and they act on the tuples stored in the tuple space depending on their properties. Thus, to propagate an information through the network, it suffices to inject a tuple containing that information and a property “SPREAD”.

Each mobile device contains a tuple space, and stored information is local to each node. It is the Spreading service that propagates the information between nodes.

We developed the different services shown in Figure 3. In each node and depending on local conditions, the middleware adapts parameters of core services (e.g. Spreading more or less quickly, or to more or less neighbours) and switches algorithms (e.g. switching from a location based implementation to probabilistic depending on the availability of neighbouring nodes’ position information).

Detailed information about the architectural design and implementation of the SAPERE middleware can be found in [14, 39].

4.2 Spreading service - Implementation

We focused the optimisation of propagation in two different directions: (1) to reduce the number of nodes that re-send the information by a given probability; and (2) to reduce the number of nodes that resend the information based on their positions.

We describe first three baseline schemes towards which we later compare our results: Pure, Probabilistic, and Location-based Spreading as described in [26]. We then propose two new propagation schemes: Adaptive probabilistic, and Switching propagation.

Pure Propagation, also called pure flooding or blind flooding is a basic algorithm for propagating information. Each node that receives the propagated information for the first time re-sends it to all neighbouring nodes.

Pure propagation produces a high number of messages, often resulting in network overload in addition to collisions, contention, and redundancy.

Probabilistic propagation. In order to reduce the number of messages, consequently reducing the chance of contention and collision, an intuitive approach is to govern the decision as to whether or not to send a message with a specified probability [32].

Notice that the same probability may not work for all different scenarios, i.e., scenarios with very low density of nodes would need to propagate information with higher probability than scenarios with high density of nodes.

Location-based propagation is a family of algorithms that take into account the position of neighbouring nodes in order to decide which nodes should re-send information. The main idea is to reduce the number of nodes that re-send the information based on the nodes’ positions.

Location-based approaches deliver good results compared with pure and probabilistic schemes, however, the computation carried out at each node is more complex, and relies on nodes providing their positions accurately. Location-based implementations can be extended by taking into account neighbouring nodes of 2 hops distance, further optimising the reduction in the number of messages sent, but increasing the computation at each node.

Adaptive probabilistic propagation is an extension of probabilistic propagation where the probability is a adapted on-the-fly by the middleware, depending on contextual information (i.e. number of neighbouring nodes).

In this implementation of the adaptive probabilistic broadcast algorithm the function that determines the probability of propagation was optimised off-line. The optimisation process aimed to reduce the number of messages while keeping all nodes received the information.

The function that adapts the probability is define as follows:

$$f(x_t) = \left(\frac{-0.2x_t}{\alpha}\right) + 1.05$$  \hspace{1cm} (1)

Where, $x_t$ is the number of neighbouring nodes at time $t$. $\alpha$ is the tuning parameter to change the slope of the function (fixed to 8). Finally, $f(x_t)$ is the resulting probability that will be applied at each node at time $t$.

Each node computes a different propagation probability depending on the contextual information (i.e. number of nodes) at each time $t$.

Switching propagation is the result of combining adaptive probabilistic and location-based propagation. Namely, we add a self-management policy that switches between these two algorithms depending on the availability of positional information.

4.3 Policies

We present here the policies that the middleware applies when managing the Spreading service. They are as follows:

Policy 1: If neighbouring nodes have access to location information, the middleware switches to a location-based approach for propagating the information.

Policy 2: If neighbouring nodes do not have location information, the middleware uses a probabilistic approach.

Policy 3: In case of adaptive probabilistic approach, the probability of propagating the information at each nodes is adapted based on the number of neighbouring nodes. This probability follows the Equation (1) calculated off-line and based on simulation results.

5. PROOF OF CONCEPT

In this paper, as a proof of concept to validate our approach to self-managing and self-organising services,
we implemented and evaluated the Spreading service (as described above), we then implemented the Gradient Service that uses the Spreading service, and finally the Chemotaxis Service using the Gradient service. In the results below, only the Spreading service is self-managed by the use of policies.

Major aspects that we highlight are: (1) it has been demonstrated in the literature (see [26]) and by our results that the parameters of the different propagation algorithms can cause failures if they are not tuned to the environment conditions; (2) the dynamic adaptation of parameters reduces the probability of failure and improves performance (adapting the probability of the probabilistic propagation according to the density of nodes); (3) some algorithms perform better than others, but require information that is not always available (e.g. position information for location-based schemes), hence the need to switch from one algorithm to another depending on available context; finally, (4) a Spreading service should be able to provide an efficient implementation to higher level services or applications independently of the environmental conditions.

These preliminary experiments use The ONE-SAPERE simulator, the extension of The ONE simulator where each node runs an instance of the SAPERE middleware. The main setting of the simulations are: a bi-dimensional space (500m x 500m), a number of nodes between 150 and 600 moving at 2-4m/s, a communication range of 80 meters, and a random walk mobility pattern. The simulator uses the actual SAPERE middleware and code in each node. What is simulated are the nodes movements and the wireless communication between the nodes.

Figure 4 shows the number of messages used to create and keep updated a Gradient, when that Gradient service uses different propagation schemes provided by the Spreading Service. The worst results regarding the number of messages are provided by the Pure propagation (i.e. no policies are applied and every node broadcast the information). The best results are provided by both the Adaptive propagation using the policy that adapts the probability depending on the density of nodes, and also by Switching propagation as it combines all cases and exploits all policies discussed above.

In this experiment only 50% of nodes provide location information. Thus, the Location-based algorithm does not outperform either the Adaptive propagation nor probabilistic propagation.

Figure 5 shows the accuracy of the dynamic gradient at each simulated second. The accuracy is the percentage of nodes (x100) that are properly updated, thus, accuracy equal to 1 implies a perfect gradient. We observe that the variants of the Spreading service that are less efficient (in terms of message consumption) naturally support the Gradient service in reaching higher accuracy. Through self-management we improve the number of messages used to build and maintain a Dynamic Gradient. This involves a slightly deteriorated quality in the accuracy of the Gradient (at each point in time and in average).

At the level of the Chemotaxis service, built using the Gradient, we observed the quality of the accuracy has no significant effect on the Chemotaxis quality. The Chemotaxis experiments involved one node sending messages to the source of the gradient. We measured the percentage of messages properly delivered and the total number of messages sent. Results showed that messages were properly delivered (100%) to the gradient source and that the number of messages needed for routing the information was similar for the different spreading implementations. We notice that the Chemotaxis service does not need a highly accurate gradient for routing information efficiently (i.e. without a significant increase in the number of messages and ensuring a 100% of messages properly delivered). Additionally, the number of messages for creating and maintaining the gradient structure is lower when self-managed spreading is used.

Figure 6 shows the propagation of a gradient among the nodes. The black lines represent the shortest paths (i.e. the paths with minimum number of hops) for reaching the gradient source.

We conclude that the self-management of the Spreading service supports the implementation of the Gradient and Chemotaxis services in an efficient way, while keeping similar performance in terms of accuracy. A major aspect of this approach is that neither the Gradient service nor Chemotaxis service are in charge of deciding how information is propagated in the system. Thus, by delegating this responsibility to the Spreading Service, separation of concerns is promoted.

6. CONCLUSIONS

In this paper we showed how self-organising mechanisms can be provided as services that other higher-level services or applications can use and rely on. We also stated that the computational environment should be in charge
of their provision as services, and guaranteeing non-functional aspects. Self-management is achieved in two ways: through seamless switching among different instances of the same service and by tuning parameters at runtime. We implemented this approach on a distributed active tuple based middleware (i.e. each mobile device contains its own tuple space) and showed how to develop efficient high-level self-organising services in a modular way that relies on lower-level services, and how to tackle two non-functional aspects: accuracy and resource optimisation. We believe that this opens a new door in engineering reliable and controllable self-organising systems.

In future works, we will consider methods to request quality of service levels, and focus on how such requirements can be propagated to and guaranteed by lower-level services. Notice that other high level services such as Morphogenesis, where the gradient value is used to estimate relative positions to one or more gradient sources, can be more sensitive to gradient accuracy, thus, research about how to request and ensure quality of spatial services is needed. Additionally, we will conduct experiments with other higher level mechanisms such as gossip, morphogenesis, and explore the use of other non-functional aspects, such as fault tolerance.

Acknowledgment

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

7. REFERENCES


H. V. D. Parunak, M. Purcell, and R. O'Connell. A catalog of biologically-inspired primitives

S. Montagna, M. Viroli, J. L. Fernandez-Marquez, M. Mamei and F. Zambonelli. Field-based motion

M. Mamei, M. Vasirani, and F. Zambonelli. A case study for self-organization: Di

D. Gaertner and K. Clark. On optimal parameters for


