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

ABSTRACT

Self-organising systems are being developed in an ad-hoc way, without reusing functionalities, thus producing a software difficult to maintain and to reuse in other applications. The development of self-organising applications and a fortiori the one of self-organising mobile applications is limited to developers who are experts in specific self-organising mechanisms. This paper discusses the notion of self-organising mechanisms provided as services for building higher level functionality in a modular way, reusing functionality and thus, providing separation of concern. 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 as well whether the self-management of self-organising mobile applications can be achieved in a modular fashion through 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 investigated two non-functional aspects: resource optimisation and accuracy.

1. INTRODUCTION

Devices with wireless communication capability and computational power such as mobile phones, tablets, and, more recently, cars are gaining the capability to build large, dense, opportunistic infrastructures, such as MANETS and 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, host heterogeneity and network segmentation.

A promising approach to architecting services for such infrastructures is provided by self-organising systems [37]. Self-organising systems consist of a set 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, 10, 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 [7]. Rather, many self-organisation techniques must be optimised for their operating environment and are not meant to overcome all possible environmental situations. Additionally, self-organising systems are being developed in ad-hoc way, without reusing functionalities, producing a software difficult to maintain and reuse in other 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 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, in addition to specified non-functional requirements, results accuracy, performance optimisation, availability or accessibility of a service.

How to engineer self-organising mobile applications in a modular way, using operator as building blocks for building higher level functionalities is still an open challenge. Additionally, if we think about builiding 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 then require self-management in order to flexibly handle dynamic environments with min-

SAC'14 March 24-28, 2014, Gyeongju, Korea.

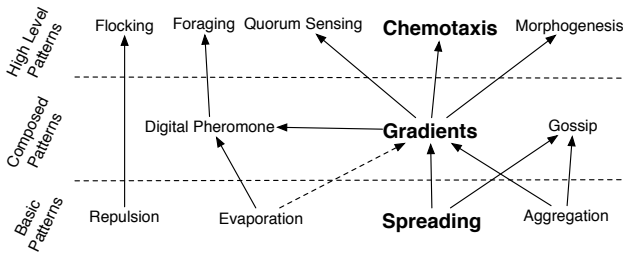


Figure 1: The Self-organising Design Patterns catalogue [15]

imal human intervention and allowing higher-level applications to reuse them.

We take a larger view here and consider it is important to separate the concerns in the following ways: (1) self-organising mechanisms - previously identified as building blocks from which higher level self-organising systems and applications can be constructed [12] - are provided as (reusable) services; (2) any application requiring the use of such a self-organising service relies on that service to self-manage and to provide non-functional aspects corresponding to the application's requirement and available contextual information; (3) a self-organising service relies on its operating environment to handle part of its non-functional behaviour.

Rather than considering the management of the complex system as a whole, this paper investigates instead whether the self-management of complex self-organising systems 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. In this paper we focus on two non-functional aspects: resource optimisation and accuracy of spatial structures in a mobile environment.

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 3.4 shows how self-management acts hand in hand with self-organisation. Section 4 discusses implementation aspects, before Section 5 shows a proof of concept and experimental results on a mobile applications, highlighting three levels of self-organising services, each level using the level below: from spreading, to dynamic gradient to chemotaxis. Section 6 concludes the paper.

2. RELATED WORK

This section briefly reviews the state of the art related to the engineering of self-organising systems in a modular way, as well as current approaches to non-functional aspects such as optimisation and adaptation to mobile conditions.

2.1 Separation of concerns in self-organising systems

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 low level mechanisms have to be implemented in order to deal with different scenarios and how they can actually be reused by applications under different environmental conditions. Pursuing this idea, Giovanna Di Marzo Serugendo et al.[8] present the notion of self-organising services actually proposing self-organising mechanisms as services that can be reused many times and by many applications. This work does not discuss self-manageming 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 self-organising systems

We start from the one of the first and most relevant self-organising systems, Ant Colony Optimisation (ACO), which was proposed in 1999 to find shortest path problems on a graph for instances of the Travelling Salesman Problem (TSP) [9]. It demonstrates that each instance of the TSP requires ACO parameters to be tuned in order to achieve efficiency. Extensions of ACO algorithms such as [17] and [38] have proposed different approaches to make the system more adaptive, able to deal with different problems without requiring human intervention to reset the parameters and perform well. Gaertner et al. [17] consider the interdependencies between parameters and conduct a correlation study which leads to a quicker convergence for parameter settings. Yoshikawa et al. [38] introduce a new type of ant called *crankyant* to prevent trapping at local optima; that is, the cranky ant explores the path which has not been selected and thus enables to change the search area. Parameter updates are based on the reinforcement of positive feedback.

Many 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 their 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 different implementations that are difficult to evaluate and compare because of their context dependent behaviours.

Some examples of extensions of these self-organising algorithm in order to improve their performance in a wider set of scenarios include: Dasgupta et al. [5] propose an optimisation for adaptive computational chemotaxis in bacterial foraging. Folino et al. [16] propose an adaptive multi-agent algorithm to cluster spatial data.

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, such as chemotaxis, flocking or morphogenesis, we first build such high-level mechanisms in a modular fashion and in the form of services. Chemotaxis is built from using a gradient and spreading service. Second, we tackle the self-management issues (performance and adaptation) by providing self-management at each level, i.e. at each self-organising service, from spreading to gradient to chemotaxis itself.

3. SEPARATION OF CONCERNS

In this section we expand on our main points for addressing separation of concerns in mobile computing applications: self-organising mechanisms provided as services; complex self-organising high-level services or applications depend on them and are modularly designed; low-level self-organising services need to exhibit functional characteristics but also non-functional characteristics as requested by the services or applications that use them; the self-management aspect is the responsibility of the operating environment. We illustrate our discussion using three self-organising mechanisms: Spreading, Gradient (using Spreading and Aggregation), and Chemotaxis relying on Gradient. These three mechanisms are also the subject of our proof of concept and experimental analysis detailed in Section 5.

The choice of these mechanisms for mobile computing applications is motivated by previous work in mobile, pervasive and context-aware scenarios [3], where spatial structures [2, 1] spread among mobile devices and maintain themselves despite the mobility or changing density of the nodes. These spatial structures are then used for developing applications such as steering groups of people in crowded areas [24], querying and retrieving information among mobile nodes.

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, 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, evaporation, etc.) 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 the three patterns Spreading, Gradient and Chemotaxis are linked to each other and how they can be designed in a modular way, and provided as services.

Spreading. A *spreading* mechanism, also called *flooding* or *propagation* is a mechanism where a piece of information is periodically sent by nodes, and eventually reaches 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.

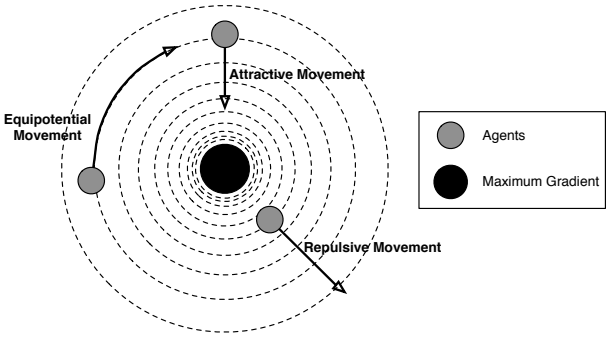


Figure 2: Chemotaxis Pattern - adapted from [6]

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 information is propagated in such a way that it provides an 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 ants 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 makes use of the Spreading service in order to diffuse itself in a distributed environment.

Chemotaxis. The Chemotaxis Pattern, initially 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 the direction of their next movements.

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, incrementing the distance between the agent and the gradient source, 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, Viroli 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.

In terms of services 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 Self-organising services: 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 regularly must ensure a certain quality of service, when dealing with dynamic environment, 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, but if not all nodes have GPS information, 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 also 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 accurately reflect the true state of the world, potentially affecting the quality of service characteristics of the 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 to allow applications to rely on it.

The trade-offs between parameter values is also highlighted in the design pattern catalogue mentioned above, 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 selection of different implementations and tuning of their parameters in order to increase the performance of the system and therefore satisfy service, application, and user requirements.

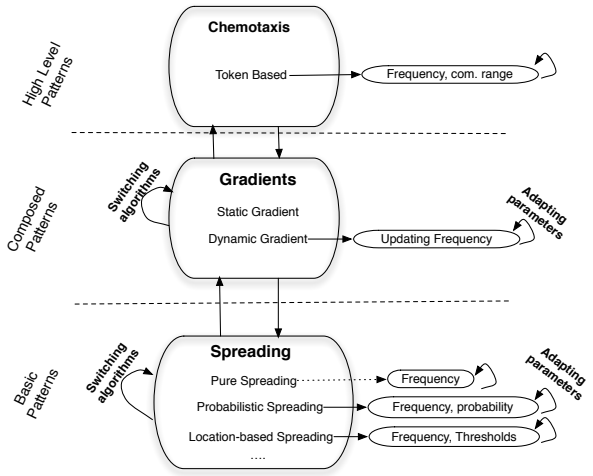


Figure 3: Self-management at different levels

3.3 Self-managing self-organising services: a responsibility of computational environments

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 that are 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 and of optimising the use of resources to guarantee the scalability of applications running on top of it.

3.4 Layered Self-Management

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 (i.e., balance) 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 with different algorithms such as probabilistic or location-based approaches, and thus provided by different instances; each instance being the optimal 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 (i.e., each implementation) 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 middleware [39], developed by the SAPERE EU Project [1].

The SAPERE middleware runs in two different 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 plays a key role in prototyping and validating new functionalites before they are released for its 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 an active 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".

Notice that each mobile device contains a tuple space, so information stored in the tuple space is local to each node, and it is the Spreading service that propagates the information among tuple spaces (i.e. among nodes).

We developed the different services shown on 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 Static Gradient when nodes are stationary to Dynamic Gradient when nodes are moving). More details about the management of the Spreading service are given below.

Detailed information about the architectural design or how the code is computed in the SAPERE middleware can be found in [14, 39]. Additional details about the work presented in this paper can be found in [4].

4.2 Spreading service - Implementation

We discuss here the different variants we implemented for the Spreading service.

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; or (2) to reduce the number of nodes that resend the information based on their positions, which requires information about their position in the system.

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 present two new propagation schemes proposed by us: *Adaptive probabilistic*, and *Switching propagation*.

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

Pure propagation produces a high number of messages involving 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 the information. The main idea is to reduce the number of nodes that re-send the information based on the nodes' positions.

Location-based approaches present very good results compared with pure and probabilistic propagation schemes, however, the computation carried on 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 dramatically the computation at each node.

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

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.

[1] http://www.sapere-project.eu/

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 an equation 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. Notice that the results below only the Spreading service is self-managed by the use of policies. We used Dynamic Gradient and Chemotaxis (token-based).

The major aspects that we highlight are: (1) It has been demonstrated in the literature (see [26]) and in our experimental results that the parameters of the different propagation algorithms can cause failures if they are not set up accordingly to the environment conditions; (2) the dynamic adaptation of parameters reduces the probability of failure, and improves the performance (adapting the probability of the probabilistic propagation according to the density of nodes); (3) some algorithms performs better than others, but they require information that sometimes is not available (i.e. Location-based scheme provides better results, but it requires all nodes to provide their position information.) Thus, the need for switching from one algorithm to another depending on the contextual information available; finally, (4) a Spreading service should be able to provide an efficient implementation to higher level services or applications independently of the environmental conditions.

In these preliminary experiments we have used The ONE-SAPERE simulator. The main setting of the simulations are: a bidimensional space (500m x 500m), a number of nodes between 150 and 600 moving at 2-4m/s, communication range of 80 meters, and a random walk mobility pattern. This simulator uses the actual SAPERE middleware and code in the nodes. What is simulated is the nodes movements and communication collisions.

Figure 4 shows the number of messages used to create and keep updated a Gradient, when that Gradient service uses different propagations 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 the Switching propagation since it combines all cases and exploits all policies discussed above.

In this experiment only 50% of nodes are providing location information. Thus, the Location-based algorithm is not performing better than the Adaptive propagation.

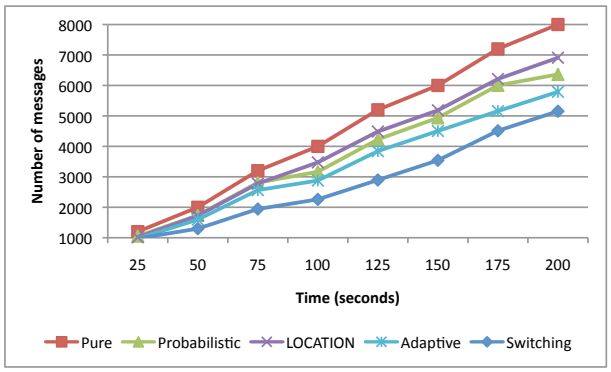


Figure 4: Gradient using different propagation algorithms - Messages

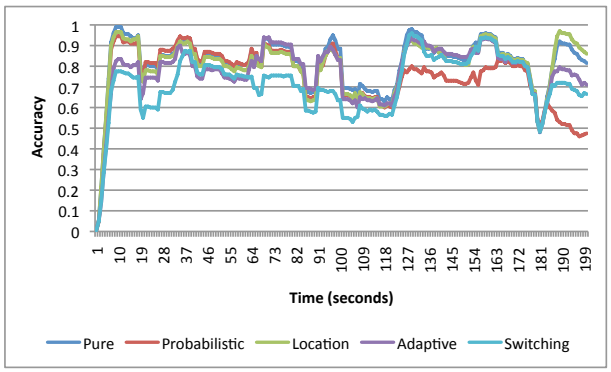
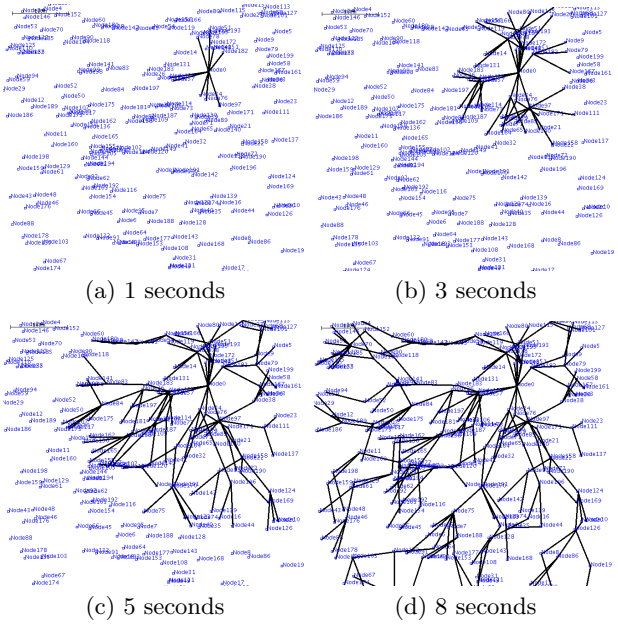


Figure 5: Gradient using different propagation algorithms - Accuracy

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 equals 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, that uses Gradient, we observed the quality of the accuracy has no significant effect on the Chemotaxis quality. Indeed, 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. Thus, the Chemotaxis service routes information efficiently while 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 path for reach-



(a) 1 seconds (b) 3 seconds

(c) 5 seconds (d) 8 seconds

Figure 6: Gradient Creation - 200 nodes

ing the gradient source.

We conclude that the self-management of the Spreading service, allows the implementation of the Gradient and Chemotaxis services in an efficient way, while keeping similar performances in terms of accuracy. A major aspect of this approach is the fact that neither the Gradient service nor Chemotaxis are in charge of deciding how the information must be propagated in the system. Thus, delegating the responsibilities on the Spreading Service.

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 needs to be in charge of both providing those mechanisms as services and guaranteeing non-functional aspects. Self-management is achieved in two ways, either through seamless switching among different instances of the same service or by tuning parameters at run-time. We implemented this approach on a tuple based middleware (i.e. each mobile device contains its own tuple space) and showed how to develop efficient high-level self-organising services (Chemotaxis and Gradients) in a modular way that relies on lower-level ones (Spreading), 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 a quality of service from lower-level services, and how this can be propagated to and guaranteed by lower-level services. We also plan to explore the use of machine learning techniques in order to allow the low level self-organising services to adapt their behaviour in order to satisfy quality requests and ensure an efficient usage of resources.

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

[1] J. Beal. Flexible self-healing gradients. In *Proc. of the 24th ACM symposium on Applied Comp.*, pages 1197–1201, 2009.

[2] J. Beal, J. Bachrach, D. Vickery, and M. Tobenkin. Fast self-healing gradients. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 1969–1975, New York, NY, USA, 2008. ACM.

[3] Blind. Blind.

[4] Blind. Techreport. Technical report.

[5] S. Dasgupta, S. Das, A. Abraham, and A. Biswas. Adaptive computational chemotaxis in bacterial foraging optimization: An analysis. *Evolutionary Computation, IEEE Transactions on*, 13(4):919–941, 2009.

[6] T. De Wolf and T. Holvoet. Design patterns for decentralised coordination in self-organising emergent systems. In *proc. of the 4th int. conf. on Engineering self-organising systems*, ESOA'06, pages 28–49. Springer-Verlag, 2007.

[7] G. Di Marzo Serugendo. Robustness and dependability of self-organizing systems - a safety engineering perspective. In *Proc. of the 11th Int. Symposium on Stabilization, Safety, and Security of Distributed Systems*, SSS '09, pages 254–268. Springer-Verlag, 2009.

[8] G. Di Marzo Serugendo and J. L. Fernandez-Marquez. Self-organising services. In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO2013)*. IEEE Computer Society, 2013.

[9] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, 1996.

[10] F. Dressler, I. Dietrich, R. German, and B. Krüger. A rule-based system for programming self-organized sensor and actor networks. *Comput. Netw.*, 53:1737–1750, 2009.

[11] J. L. Fernandez-Marquez, J. L. Arcos, and G. Di Marzo Serugendo. A decentralized approach for detecting dynamically changing diffuse event sources in noisy wsn environments. *Applied AI Journal*, 26(4):376–397, 2012.

[12] J. L. Fernandez-Marquez, J. L. Arcos, G. Di Marzo Serugendo, and M. Casadei. Description and composition of bio-insp. design patterns: the gossip case. In *Int. Conf. on Engineering of Autonomic and Autonomous Syst. (EASE)*, pages 87–96. IEEE Computer Society, 2011.

[13] J. L. Fernandez-Marquez, J. L. Arcos, G. Di Marzo Serugendo, M. Viroli, and S. Montagna. Description and composition of bio-inspired design patterns: The gradient case. In *Workshop on Bio-Insp. and Self-* Algorithms for Distributed Systems (BADS)*, pages 25–32. ACM, 2011.

[14] J. L. Fernandez-Marquez, G. Di Marzo Serugendo,

and S. Montagna. Bio-core: Bio-inspired self-organising mechanisms core. In *Bio-Inspired Models of Networks, Information, and Computing Systems*, volume 103 of *LNCS, Social Informatics and Telecommunications Engineering*, pages 59–72. Springer Berlin Heidelberg, 2012.

[15] J. L. Fernandez-Marquez, G. Di Marzo Serugendo, S. Montagna, M. Viroli, and J. L. Arcos. Description and composition of bio-inspired design patterns: a complete overview. *Natural Computing*, pages 1–25, 2012.

[16] G. Folino and G. Spezzano. An adaptive flocking algorithm for spatial clustering. In *Proc. of the 7th Int. Conf. on Parallel Problem Solving from Nature*, PPSN VII, pages 924–933. Springer-Verlag, 2002.

[17] D. Gaertner and K. Clark. On optimal parameters for ant colony optimization algorithms. In *Proc. of the Int. Conf. on Artificial Intelligence 2005*, pages 83–89. CSREA Press, 2005.

[18] D. B. Johnson, D. A. Maltz, and J. Broch. Ad hoc networking. chapter DSR: the dynamic source routing protocol for multihop wireless ad hoc networks, pages 139–172. Addison-Wesley Longman Publishing Co., Inc., 2001.

[19] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[20] A. Keränen, J. Ott, and T. Kärkkäinen. The ONE Simulator for DTN Protocol Evaluation. In *SIMUTools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, New York, NY, USA, 2009. ICST.

[21] M. Mamei, R. Menezes, and F. Tolksdorf, R.and Zambonelli. Case studies for self- organization in computer science. *J. Syst. Archit*, 52:433–460, 2006.

[22] M. Mamei, M. Vasirani, and F. Zambonelli. Experiments of morphogenesis in swarms of simple mobile robots. *Journal of Applied Artificial Intelligence*, 18:903–919, 2004.

[23] M. Mamei and F. Zambonelli. Field-based motion coordination in quake 3 arena. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '04, pages 1532–1533. IEEE Computer Society, 2004.

[24] S. Montagna, M. Viroli, J. L. Fernandez-Marquez, G. Di Marzo Serugendo, and F. Zambonelli. Injecting self-organisation into pervasive service ecosystems. *Mobile Networks and Applications*, 18(3):398–412, 2013.

[25] R. Nagpal. A catalog of biologically-inspired primitives for engineering self-organization. In *Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering*, pages 53–62. Springer, 2004.

[26] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proc. of the 5th ACM/IEEE Int. Conf. on Mobile comp. and networking*, MobiCom '99, pages 151–162. ACM, 1999.

[27] R. Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Transactions on Automatic Control*, 51:401–420, 2006.

[28] H. V. D. Parunak, M. Purcell, and R. O'Connell. Digital pheromones for autonomous coordination of swarming uavs. In *The First AIAA Unmanned Aerospace Vehivales, Systems, Technologies, and Operations*, pages 1 – 9, 2002.

[29] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing. RFC editor, 2003.

[30] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proc. of the Second IEEE Workshop on Mobile Computer Systems and Applications*, WMCSA '99. IEEE-CS, 1999.

[31] M. Puviani, G. Di Marzo Serugendo, R. Frei, and G. Cabri. Methodologies for self-organising systems: A spem approach. In *Proc. of the 2009 IEEE/WIC/ACM Int. Joint Conf. on Web Intelligence and Intelligent Agent Technology*, WI-IAT '09, pages 66–69. IEEE Computer Society, 2009.

[32] Y. Sasson, D. Cavin, and A. Schiper. Probabilistic broadcast for flooding in wireless mobile ad hoc networks. In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, volume 2, pages 1124–1130 vol.2, 2003.

[33] C. F. Tschudin. Fraglets - a metabolistic execution model for communication protocols. In *In Proceeding of 2nd Annual Symposium on Autonomous Intelligent Networks and Systems (AINS), Menlo Park*, 2003.

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

[35] D. Weyns, A. Omicini, and J. Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, Feb. 2007.

[36] B. Williams and T. Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proc. of the 3rd ACM Int. symposium on Mobile ad hoc networking & computing*, MobiHoc '02, pages 194–205. ACM, 2002.

[37] T. Wolf and T. Holvoet. Emergence versus self-organisation: Different concepts but promising when combined. In S. Brueckner, G. Marzo Serugendo, A. Karageorgos, and R. Nagpal, editors, *Engineering Self-Organising Systems*, volume 3464 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2005.

[38] M. Yoshikawa and T. Nagura. Adaptive ant colony optimization considering intensification and diversification. In S. I. Ao, O. Castillo, C. Douglas, D. D. Feng, and J.-A. Lee, editors, *Proceedings of the International MultiConference of Engineers and Computer Scientists 2009 Volume I, IMECS '09, March 18 - 20, 2009, Hong Kong*, Lecture Notes in Engineering and Computer Science, pages 200–203. International Association of Engineers, Newswood Limited, 2009.

[39] F. Zambonelli, G. Castelli, M. Mamei, and A. Rosi. Integrating pervasive middleware with social networks in SAPERE. In *Int. Conf. on Selected Topics in Mobile and Wireless Networking*, 2011.