Is self-expression useful? Evaluation by a case study

Mariachiara Puviani*, Carlo Pinciroli†, Giacomo Cabri*, Letizia Leonardi* and Franco Zambonelli*
* Università degli Studi di Modena e Reggio Emilia, Modena, Italy
Email: name.surname@unimore.it
†IRIDIA, Université Libre de Bruxelles, Bruxelles, Belgium
Email: cpinciro@ulb.ac.be

***Abstract*—In the context of adaptive component-based systems, self-expression is the capability of changing the adaptation pattern at run-time when some changes occur in the system itself or in its environment. Even if functional requirements can be met without changing the adaptation pattern, the achievement of non-functional requirements, such as performance, can benefit from a change of adaptation pattern. The aim of this paper is to show, by means of a case study in swarm robotics, that a change of adaptation pattern can affect the performance of a system. In particular we show that the change of adaptation pattern can be useful in adaptive component-based systems to react to changes in the situation of the environment.**

***Keywords*—Self-expression, adaptation, pattern, swarm, robots, ARGoS.**

I. Introduction

In software engineering, *adaptation* is defined as the ability of a system to autonomously adapt its behaviour to dynamic operating conditions [1]. Software systems need to adapt to failures, changes in their computing and physical environments, and to the availability of new or upgraded services [2].

Adaptation can occur at two levels: at the level of a single *component*, and at the level of the entire system viewed as an *ensemble* of components. The first level consists in explicitly programming each component as adaptive; while the second one is based on the idea that multiple components must be able, as an ensemble, to exhibit an adaptive behaviour even if the single components are not all adaptive. This is possible when there is a component (e.g. an *autonomic manager*) that is able to manage the system adaptation by means of an explicit feedback loop or when the interaction of components is such to implement the feedback loops. To this purpose the adoption of an architectural pattern for self-adaptation (simply called "adaptation pattern" in the rest of the paper) can be very useful because it describes how to engineer an adaptive system by properly structuring its feedback loops.

The self-* properties (self-healing, self-configuring, self-optimizing, self-protecting and self-adapting) act in a system at the level of a single component, while self-expression acts at the level of the entire system. When changes influence all the components of the system, the self-* properties are not enough to "re-organise" the entire system and maintain a sufficient level of its performances. So a new reconfiguration of the system is necessary. This is what we call "self-expression" [3]: a meta-level form of structural adaptation. We mean that a system can "express" in autonomy the most suitable adaptation pattern in response to unexpected situations. It concerns the adoption of a new adaptation pattern to change the configuration of the system: it is a sort of adaptation of the used adaptation pattern.

The contribution of this paper is to show the usefulness of the concept of self-expression. To this purpose, we apply our ideas to a targeted swarm robotics scenario, the foraging: robots must collect food items. With this case study, we experiment that the pattern chosen for the system creation can exhibit some limits at runtime, when the environment and/or system conditions change. While in an open arena a pattern based on stigmergic interactions performs better, a pattern with a centralised manager better suits in the presence of obstacle. Therefore, we show that applying self-expression makes the system maintain acceptable levels of performances also when changes occur.

The paper is organised as follows: in Section II we introduce the concept of adaptation patterns and self-expression. Then, we present the case study we used to evaluate self-expression (Section III). The main contribution of the paper, i.e. the results of the simulations along with their explanation, is presented in Section IV. Finally, in Section V we conclude the paper and sketch possible directions for future work. Here we do not present any related works because, as far as we know there is no work comparable to our.

II. Adaptive architectural patterns and self-expression

In the literature design patterns [4] (or simply patterns) are defined as reusable solutions to recurring design problems and are a mainstream of software reuse practice [5]. They crystallize a general solution to a common problem, so software developers can benefit from their reuse to develop systems.

An *adaptation pattern* for complex distributed systems is a conceptual scheme that describes a specific adaptation mechanism. It specifies how the component/system architecture can express adaptivity. For example, the *Reactive Stigmergy Pattern* describes how, in a system with a large number of simple components, adaptation is enacted using signals in the environment as the only communication mechanisms [6].

Unfortunately nowadays it is not possible to immediately recognize which is the most appropriate pattern for the problem under study. Thus, it is useful to have guidelines that enables developers to choose the one they think is the most appropriate for their needs. Therefore we created a catalogue of self-adaptation pattern presented in [7]. In this catalogue, we propose a list of adaptation patterns described using a template, and we offer some examples for their use in order to choose the right one.

978-0-7695-5002-2/13 $26.00 © 2013 IEEE
DOI 10.1109/WETICE.2013.46

Once a specific pattern is applied to a system, this system will exhibit an adaptive behaviour i.e. is capable of delivering its functionalities despite changing conditions. Nevertheless, at runtime something can change in the system or in its environment. Thanks to its adaptive behaviour, the system keeps meeting its *functional* requirements, but the changes can affect the *non-functional* requirements, for instance the performance can decrease because the conditions for having chosen such pattern do no longer hold. In this case, when the application of self-* properties are not enough, a different adaptation pattern can better meet the *non-functional* requirements than the pattern chosen for the development of the system at the beginning.

So, when a pattern adopted in a system appears to be no longer adequate to properly and/or effectively support adaptation, a re-engineering of the structure of the system may be required, to change the adaptation pattern [7]. Changing the structure of the system by means of the selection of a new adaptation pattern, that is what we call "self-expression", makes it possible to maintain the same performances in the considered system.

III. A SWARM ROBOTICS CASE STUDY

Nowadays robots are at the down of a new era, becoming helpers that improve our quality of life by delivering efficient services in homes, offices, public places and so on [8], [9], [10]. For example they can be used for collective construction, emergency rescue, and manufacturing work. This implies to enable robots to dynamically adapt in the environment where they live.

From a software engineering perspective, *swarm robotics* represents a good example of component-based ensemble. For our purposes, swarm robotics is an ideal starting point to understand how self-adaptation and self-expression work.

In swarm robotics, task allocation [11] (i.e. the activity of assigning robots to a specific task) is a well-known problem ([12], [13]). An application scenario to task allocation is *foraging*, that refers to the activity animals conduct to find, collect and harvest food. In foraging, the task of the robots is search for objects (e.g. pieces of wall or the so called food items) and transport them.

An example of this scenario is a system where the *goal* of each robot is to search for food items placed in a square area, and bring them back to the nest (each robot can carry at most one item). Each robot has also the *sub-goal* of avoiding obstacles (e.g. walls, other robots or objects) on its way. Thus, each robot needs to change its route in order to avoid obstacles while adapting its behaviour to a diminishing number of available food items. Robots must adapt because their energy consumption affects the nest energy. The "nest energy" is an experiment-specific abstraction used in the simulation scenario in order to evaluate the system's performances. As reported in 1 the nest energy (E_{Tot}) is the sum of the energy of each food item bring in the nest ($E_{retrieval}$), where O is the number of collected objects, and of the energy consumed by each of N robots while working outside the nest ($E_{consumed}$). The goal of the system is to increase the nest energy. This energy decreases during the execution, due to the energy consumption of each robot. At the same way every items arrived in the nest makes the energy level increase.

$$E_{Tot} = \sum_{o=0}^{O} E_{retrieval} - \sum_{r=1}^{N} E_{consumed} \tag{1}$$

The main *constraint* of each robot is to avoid running out of batteries. Navigating outside the nest, robots are consuming energy taken from the nest itself. Each robot must save its energy in order to keep the whole system up and running. Due to space limitation, we use a scenario that is quite simple but that well suits our aim.

In order to simulate the robot behaviour for the foraging case study, we use ARGoS (http://iridia.ulb.ac.be/argos/). ARGoS is a state-of-the-art, open source robot simulator focusing on the simulation of the physics of large heterogeneous robot swarms [14].

We define an arena where robots are free to move, and a nest (see Figure 2) where the robots rest.

To fully understand how robots work, it is important to consider the sensors and actuators available on the robots. In particular, we use the *wheel speed sensor* and *actuator* to read and modify the wheel speed in order to change the robot direction. Then, the *infrared proximity sensors* positioned on the base of the robot are used to detect obstacles. The *light sensors* are used to detect any light source in the environment that in our case indicate the position for the nest. Then the *ground sensors* are useful to detect the colour of the ground in order to recognize the nest (coloured of grey in our example). The *LED actuators* are used to control the colour and the intensity of each LED distributed around a robot (LED ring). Then a robot uses its actuators as "effectors", to propagate information into the environment.

IV. SIMULATIONS

We designed a set of simulating experiments to analyse and assess the performance of an adaptive solution for our foraging scenario. In particular, our adaptive solution is based on the Reactive Stigmergy Pattern [15]. We choose this pattern because it is the one that better suits what the system has to do [16].

This pattern, reported in Figure 1, describes the system composed of a large amount of components. These components can be extremely simple; they do not need to be able to exchange information, but they use the environment to reach information. They adapt in reaction to the information collected, and they implicitly coordinate one to each other using this information. In this pattern, the system's goal is achieved thanks to the adaptive behaviour that each component exhibits. The group behaviour results from the interaction among individuals. Communication between robots is achieved using the *environment* (information propagated in it) as ants leave pheromone in the environment and are able to sense the one let from others ants. This implicit coordination between robots permits the system to obtain the global goal and to adapt the group behaviour.

We use this pattern to implement the system's simulation described in the following.

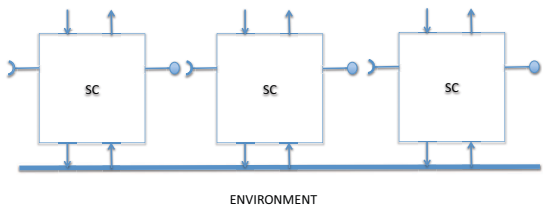

Fig. 1. Reactive Stigmergy Service Component (SC) Ensemble Pattern

A. System configuration

In our simulations, the robots move in a square arena, in which a set of food items is uniformly distributed (see Figure 2). We represented the objects as black dots on the ground. When a robot navigates upon a black dot, the dot is removed from the ground and it is automatically assigned to the robot. The grey area in the figure represents the nest where robots must bring the food, and where they can recharge their batteries. When a robot has a food item or its battery level is low (under a certain threshold), it needs to go to the nest. To do that it uses its light sensors to identify the lights that indicate the nest (see the lower part of Figure 2). While searching for food, robots use proximity sensors to avoid obstacles (e.g. other robots, walls, objects), and ground sensors to recognize the food items.

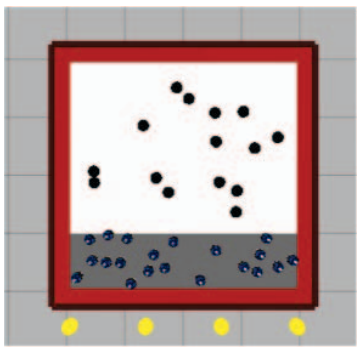

Fig. 2. Experimental arena: white floor, gray nest, food items represented as black dots and robots in the nest.

At each time step, a robot can be in one of the following state:

- $STATE\_RESTING$
- $STATE\_EXPLORING$
- $STATE\_RETURN\_TO\_NEST$
- $STATE\_OUT\_OF\_ENERGY$

Transitions among these states are probabilistically determined (see Figure 3). For example, when a robot is exploring, its probability to return to the nest gradually increases because batteries are discharging. On the contrary, the probability to explore increases if food is abundant around the robot. It is not always possible for all the robots to explore. In particular, when the level of the nest energy is not enough to recharge all their batteries, the robot's probability to explore decreases.

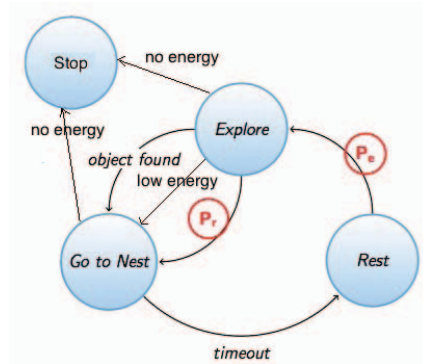

Fig. 3. State automata modeling robot behavior

None of the robots has access to the complete state of the system and they do not communicate directly. The changes in probabilities are propagated using the environment: each robot updates its status and its probabilities into the environment using its effectors. The change of probabilities models the behaviour of each robot and makes their state to change. Every robot, in order to satisfy its goal to collect food, changes its probabilities and with them its behaviour. Since these changes are propagated by the environment, other robots can sense them and so each robot adapts its behaviour. Doing that they change probabilities again. That makes the system continuously adapt its behaviour to the change of probabilities.

Every time a robot finds an item, the latter is captured by that robot. Only when the robot drops the item into the nest, a new food item appears into the arena (using a uniform distribution).

The nest energy must be preserved as long as possible. Every time a food item reaches the nest, the energy is increased by a specific value that represents the energy contained in the food item. So, the more food items are carried to the nest, the more the total level of energy is increased. Instead, every time a robot needs to go out of the nest, its battery is fully recharged, and this energy is taken from the nest. While the robot is running outside the nest looking for food, it consumes 1 energy unit per second. When its battery level goes under a certain threshold (e.g. 20% of its energy), the robot decides to come back to the nest in order to avoid battery exhaustion.

In [6] we present some different simulations performed using the Reactive Stigmergy pattern. With these simulations, we confirmed that for specific *functional* and *non-functional* requirements of the case study, the applied pattern is a suitable choice.

In the following we show some simulations to evaluate if the chosen pattern is suitable also for totally different environmental conditions. In the presented simulations, we use a fixed number of robots (e.g. 20).

B. Simulation: changing number of obstacle

In this simulation, we show how the system of robots adapts itself when the number of obstacles in the environment changes.

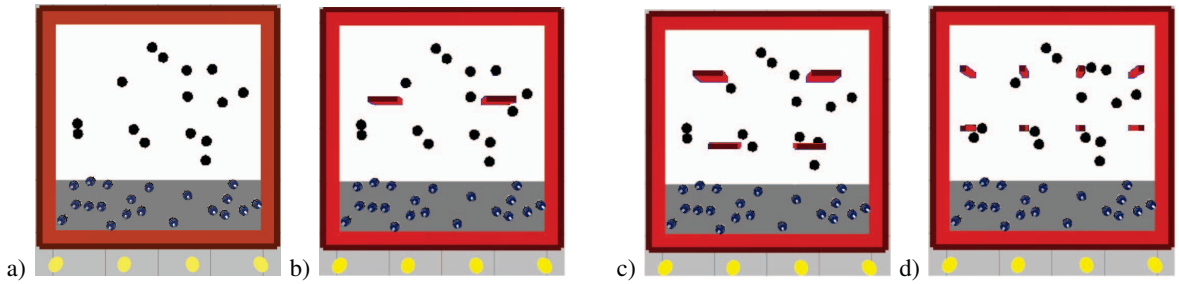

Fig. 4. Simulation arenas with no, 2, 4, and 8 obstacles.

Figure 4 shows the experimental settings we used: with no obstacles, with 2 obstacles, with 4 obstacles and with 8 obstacles.

We can see from Figure 5 (a), that the number of walking robots - explorers - is quite the same for the first part of the simulation (for around 700 sec). After that time, the number of robots walking within the arena with more obstacles sharply reduces. This happens because it is more difficult to find food and to come back to the nest, due to the obstacles to avoid. So robots spend more energy to avoid obstacles and their probability to stay in the nest rapidly increases.

The same considerations can be done looking at the collected food (see Figure 5 (b)) in the different settings. The number of food items that are brought to the nest is less when there are more obstacles in the arena.

Analysing the behaviour of this system developed using the "Reactive Stigmergy Pattern" we find out that this pattern is not the most appropriate one in certain conditions (presence of obstacles) because the system still continues to adapt, but it performances (*non-functional* requirements) decrease.

C. New pattern application

Looking at the changes in the environment and at the non-functional requirements of the system with obstacles, we find out in the catalogue of adaptation patterns [7] that the "Centralised Autonomic Manager Pattern" can help in our case.

The choice of the new pattern is now done manually, but we are implementing self-expression mechanisms that will make the choice and the change of the pattern in an autonomic way.

The pattern, presented in Figure 6, is centred around a unique feedback loop. All the components are managed by a unique Autonomic Manager (AM) that "controls" all the components behaviour and, by sharing knowledge about all the components, is able to manage adaptation in the system.

However, in this case study, something that acts as an external AM is not present. In general, the possibilities to change the structure of the system are two: (i) one robot can assume the role of manager (but this requires that the robot is intelligent enough to coordinate other robots), or (ii) there is an external component that can easily interact with the system. In our case an external camera can assume the role of an AM, and can directly interact with every single robot without changing their internal structure.

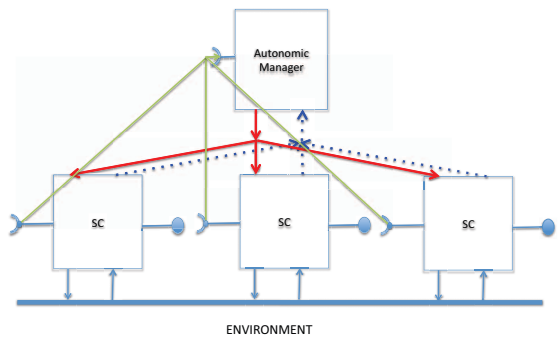

Fig. 6. Centralised AM Service Component Ensemble Pattern

The camera is positioned over the arena and it is able to direct the robots' route. This allows each robot not to spend too much time trying to avoid obstacles. It is not important that the camera knows the position of food items to increase the performances of the system; it is enough that the camera knows the obstacles position. The camera is able to find out when a robot is approaching an obstacle, and makes it change its route. The camera communicates with the robot using infrared.

In Figure 7, we can see the comparison between the systems using the two different patterns. In the arenas with obstacles (see Figure 7 (b) - (c) and (d)) the performances of the system that uses the Reactive Stigmergy pattern, start to decrease at the time of 300 sec, while using the Centralised AM pattern the performances remain high in presence of obstacles.

The application of this pattern is possible because the number of robots is not large, and so the centralised AM is not considered a single point of failure. However, if the camera fails for a short time, the robots are able to continue their task. Otherwise it the camera fails for long time, another adaptive pattern needs to be applied.

It is important to remark that, in an environment without obstacles, the performances of a system developed using the Centralised AM pattern are not as good as the one using the Reactive Stigmergy pattern, as we can see from Figure 7 (a). This because the receiving of information from the AM delays the system when there is no need for it (no obstacles).

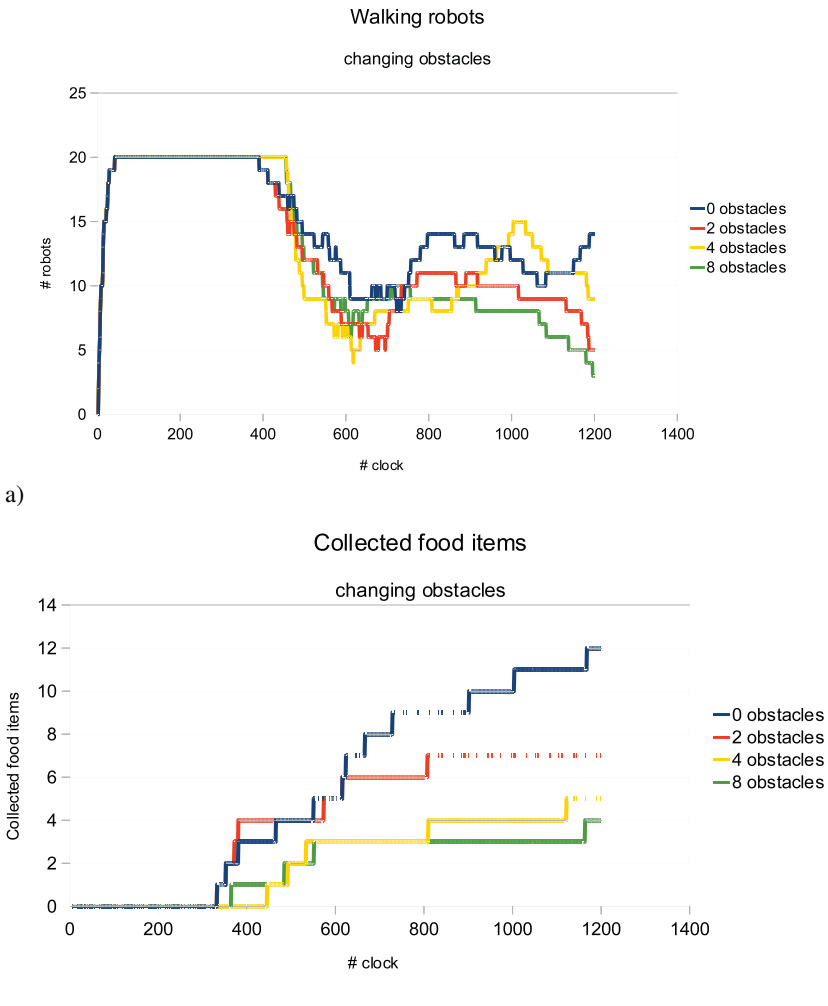

Fig. 5. Walking robots (a) and overall collected food items (b) using the stigmergy pattern

V. Discussion and Conclusions

The foraging case study presented in this paper allows us to show that, using the right adaptation pattern, the behaviour of a system of robots can be dynamically and autonomically adapted to different environmental conditions (e.g. obstacles).

Using an appropriate adaptation pattern makes it possible to obtain an adaptive system also starting from components that behave simply in a probabilistic way (like robots) and that have a little amount of information about the environment and others components.

However, the experimental simulations show that some changes at runtime can require to reorganise the system structure. This is possible by dinamically changing the adaptation pattern used to implement the system (i.e. by applying self-expression).

From the point of view of the system functional requirements, the Reactive Stigmergy pattern can be adopted to create an adaptive system that continues to be adaptive in spite of the changes in the environment. However in presence of obstacles, the way to reach the system's goal changes, and non-functional requirements, as performances, could not be met in the same way. So, in these cases self-expression is needed to adaptively change the structure of the system (e.g. adding an external camera) to reach the system's non-functional requirements.

To support the usefulness of the self-expression mechanism, we performed other simulations (not reported here due to space limitation) changing the number of robots during the system life. In the experiment with a small number of robots or a large number of robots (small/large with respect to the number of food items), the performances of the system developed with the Reactive Stigmergy pattern start to decrease. Therefore also in these cases, self-expression is useful to meet non-functional requirements.

Our future work will focus on enabling self-expression, not only creating a table that describes how patterns can be changed into others, but also in describing the mechanisms of how it is possible in concrete to migrate from an adaptive pattern to another. We are currently trying to apply self-expression mechanisms using "code migration" or alternatively

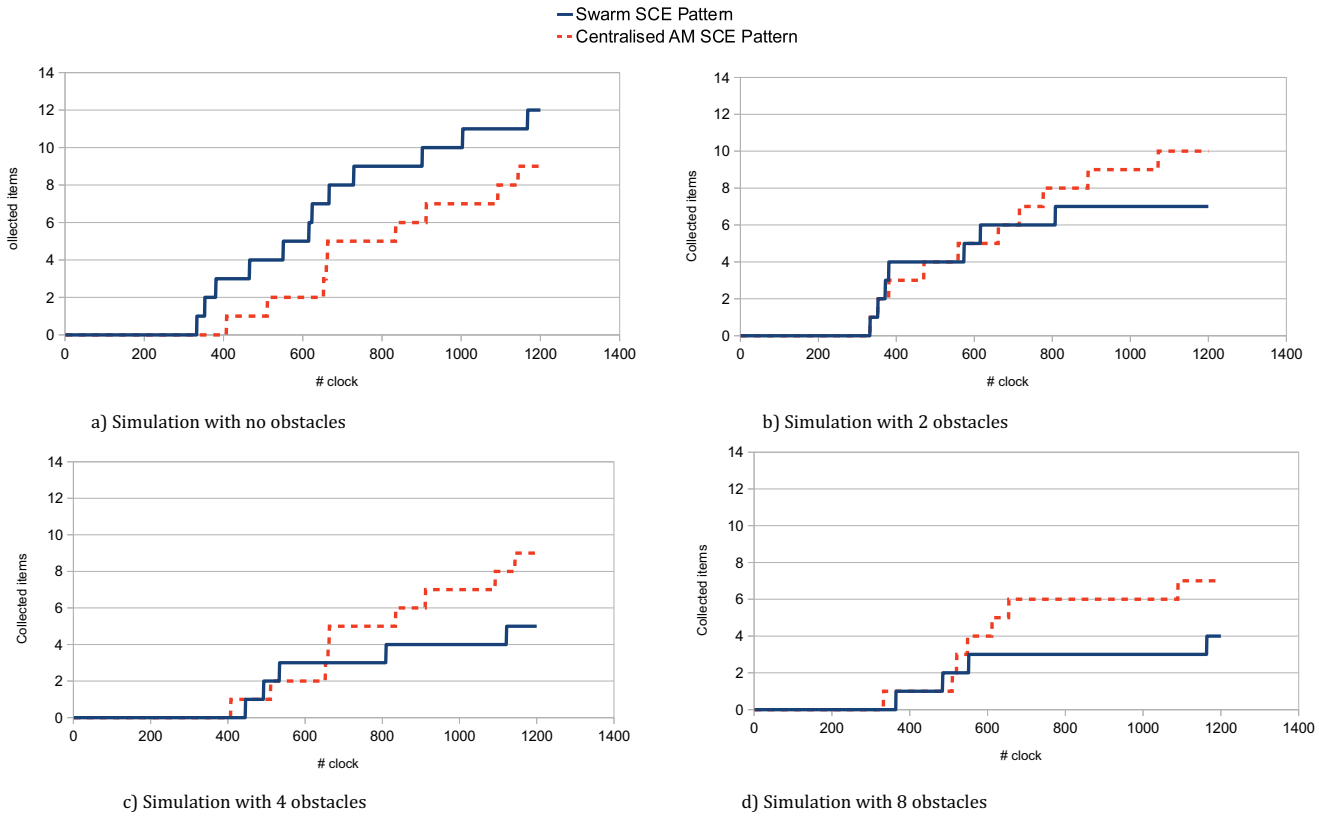

Fig. 7. Comparisons of the performances of the two different patterns.

"role assignment". This will enable to select and apply the pattern in an autonomic way.

Acknowledgment

Work supported by the ASCENS project (EU FP7-FET, Contract No.257414).

References

[1] D. Weyns and T. Holvoet, "An architectural strategy for self-adapting systems," in *Software Engineering for Adaptive and Self-Managing Systems, 2007*. Minnesota, USA: IEEE Computer Society, 2007, p. 3.

[2] G. Edwards *et al.*, "Architecture-driven self-adaptation and self-management in robotics systems," in *Software Engineering for Adaptive and Self-Managing Systems, 2009*. Vancouver, BC, Canada: IEEE, 2009, pp. 142–151.

[3] F. Zambonelli, N. Bicocchi, G. Cabri, L. Leonardi, and M. Puviani, "On self-adaptation, self-expression, and self-awareness, in autonomic service component ensembles," in *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011*. Ann Arbor, Michigan, USA: IEEE, 2011, pp. 108–113.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison Wesley, Reading (MA), 1995.

[5] M. Morandini, L. Sabatucci, A. Siena, J. Mylopoulos, L. Penserini, A. Perini, and A. Susi, "On the use of the goal-oriented paradigm for system design and law compliance reasoning," in *iStar 2010–4 th International i* Workshop*, Hammamet, Tunisia, June 2010, p. 71.

[6] M. Puviani, "Adaptation in the robotics case study: Early simulation experiences," ASCENS Project, Tech. Rep., 2011.

[7] F. Zambonelli, D. Abeywickrama, G. Cabri, M. Puviani, M. Hoelzl, A. Corradini, A. Lafuente, and R. De Nicola, "D4.2.0: Second Report on WP4," ASCENS Project, Tech. Rep., 2012.

[8] J. Inglés-Romero, C. Vicente-Chicote, B. Morin, and O. Barais, "Towards the automatic generation of self-adaptive robotics software: An experience report," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2011*. IEEE, 2011, pp. 79–86.

[9] M. Puviani, "Self-expression in adaptive architectural patterns," *Awareness Magazine: Self-awareness in autonomic systems*, 2012.

[10] J. Werfel and R. Nagpal, "Extended stigmergy in collective construction," *Intelligent Systems, IEEE*, vol. 21, no. 2, pp. 20–28, 2006.

[11] M. Krieger and J. Billeter, "The call of duty: Self-organised task allocation in a population of up to twelve mobile robots," *Robotics and Autonomous Systems*, vol. 30, no. 1, pp. 65–84, 2000.

[12] B. Gerkey and M. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[13] M. Frison *et al.*, "Self-organized task partitioning in a swarm of robots," *Swarm Intelligence*, pp. 287–298, 2010.

[14] C. Pinciroli *et al.*, "ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*. Los Alamitos, CA: IEEE, 2011, pp. 5027–5034.

[15] M. Puviani, "Catalogue of architectural adaptation patterns," ASCENS Project, Tech. Rep., 2012.

[16] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*. Oxford University Press, USA, 1999.