www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

D4.4: Fourth Report on WP4

Grant agreement number: **257414**
Funding Scheme: **FET Proactive**
Project Type: **Integrated Project**
Latest version of Annex I: **Version 3.0 (29.4.2014)**

Lead contractor for deliverable: **UNIMORE**
Author(s): **Nicola Bicocchi, Nicola Capodieci, Victor Noel, Franco Zambonelli (UNIMORE), Petr Tůma, Tomáš Bureš, Vojtěch Horký (CUNI)**

Reporting Period: **4**
Period covered: **October 1, 2013 to March 31, 2015**
Submission date: **March 12, 2015**
Revision: **Final**
Classification: **PU**

Project coordinator: **Martin Wirsing (LMU)**
Tel: **+49 89 2180 9154**
Fax: **+49 89 2180 9175**
E-mail: **wirsing@lmu.de**

Partners: **LMU, UNIPI, UDF, Fraunhofer, UJF-Verimag, UNIMORE, ULB, EPFL, VW, Zimory, UL, IMT, Mobsya, CUNI**

SEVENTH FRAMEWORK PROGRAMME

Executive Summary

In this report we summarize the work performed in WP4 during the fourth year of the ASCENS project, and the key results achieved. First, we frame the overall research approach that we have adopted. Then we present our progresses in the definition of a methodology for engineering emergence, and show how this has been applied to the robotics case study. We also show how we have identified a model to express self-expression patterns in SCEL. Finally, describe the final implementation of the architecture we have realized to support self-aware patterns, and the innovative virtual sensors we have realized within.

Contents

1 Introduction and Research Approach

The main focus of WP4 for the fourth period of the ASCENS project has been to finalize and assess the various models and tools developed over the first periods. These include the methodological tool for engineering emergence, the models for reasoning with and programming self-adaptation and self-expression, the architectures for self-awareness and its components, and the performance awareness tools. assess (via implementations and simulation experiments) the quality of the pattern catalogue produced in the second year.

1.1 Key Contributions and Research Approach

The activities of the previous three years have led to the achievement of the following research results, already extensively discussed in D4.1, D4.2, D4.3, and D4.5 (and published in international journal and conferences):

- **SimSOTA.** Finalized and assessed implementation of the SimSOTA tool, which provides tested implementation templates for all the major self-adaptive patterns in the catalogue, and that has been extensively used in the context of the e-mobility case study.
- **Patterns.** Finalized and assessed definition of the catalogue of self-adaptive and self-expression patterns, with extensive experimentation on the ASCENS case studies via both simulation and agent-based prototyping, there include the representation of the patterns in SCEL.[1]
- **Methodology.** Initial analysis of the methodological challenges in engineering ensembles that exploit those patterns in the catalogue based on self-organizing emergent behaviors, and initial identification of the best methodological approaches for possible adoption in the ASCENS project.
- **Architecture.** Implementation of a flexible architecture to support the self-aware execution of self-adaptive patterns, and its experimentation on specific classes of problems related to mobility and robotics;
- **Performance.** Definition of multiple mechanisms and tools for performance awareness, i.e., tools for the collection of performance data with high accuracy and low overhead, for the analysis of the collected data and detection of significant changes, and for the necessary adaptation in response or in anticipation of such changes.

Given that the activity related to SimSOTA were terminated, the novel contributions that have been provided in the fourth period towards the completion of the overall WP4 framework and its full integration in the ASCENS one are related to the completion of activities 2, 3, 4, 5, and include:

- **Patterns.** The identification of a novel approach to model and implement self-expression patterns, that is, modeling them in terms of holonic multiagent systems. The results of this work naturally continues the activity on prototyping self-adaptive patterns in agent-based terms, and perfectly integrates with WP2.

[1] We are aware that the self-expression patterns, as currently defined, may still suffer from not having properly addressed the issues of the possible inconsistencies that may arise during the switch from a pattern to another, and may require specific solutions for distributed agreement and leader election. At the same time, though, by adopting the SCEL language, the fact that ensembles share a knowledge base (i.e., access a common ensemble-level tuple space), and uses such knowledge base to determine the very structure of the ensemble, mitigates the potential emergence of inconsistencies.

- **Methodology.** The finalization of the methodological approach that has been defined for approaching the issue of engineering emergent behaviors in large-scale ensemble, and its exemplification on the swarm robotics case study.
- **Architecture.** The integration within the self-awareness architectures of additional sensor and classifier components, specifically conceived for adoption in e-mobility and swarm robotics scenario, and the integration of the architecture with the KnowLang approach of WP3.
- **Performance.** We have achieved some innovative contributions related to robust methods of change detection. In particular, we have defined a new interpretation of the performance comparison operators of the Stochastic Performance Logic that addresses the issues of robustness.

1.2 Relations with other WPs

The adopted research approach clearly implies a strict coordination with other WPs, and helps positioning and relating with respect to them. In particular:

- The cooperation with WP2 already started in the previous periods, has continued with the goal of analyzing how to express in SCEL also the parts related to the modeling of self-awareness patterns;
- WP4 has continued strictly cooperating with WP3, and for this latter period such cooperation has involved integrating the self-awareness architecture with the KnowLang approach developed within WP3;
- All activities have been performed, as from previous years, by focusing on the practical application scenarios, as being studied in WP7.

The overall integration of the WP4 activities in the ASCENS project is also extensively testified by their role in the ensemble software engineering lifecycle, as described in [BDG+13].

1.3 Structure of the Document

The remainder of this document is organized as follows:

- Section 2 report on the representation of self-adaptive patterns in terms of holonic multiagent systems. The results of this work are described in more details in [CCZ14]
- Section 3 presents the final methodological approach to engineering emergent behaviors in large-scale ensemble. More details on this approach have been presented in [NZ14a] and [NZ14b], and a completed extended version of the work will be include as a chapter in the forthcoming ASCENS book.
- Section 4 describe the implemented architecture that enable to enforce a peculiar self-expression pattern in the acquisition of knowledge by an SC (or SCE). The results of this work are extensively described in [BFZ14b] and [BVZH14], other in the forthcoming ASCENS book.
- Section 5 describes the progresses over the the performance awareness approaches previously described in D4.5.

Eventually, Section 6 summarizes.

2 Self-Expression Patterns and Holons

Self-Expression [ZBC+11] can be considered a peculiar Self-* property that ensembles of autonomic components can exhibit [PPC+13]: let us suppose to have a set of computational units (they may be software agents, robots or anything that can show even a small degree of adaptivity) that has to collaborate in order to solve a task. This task is a computational problem that requires a system transition from a certain initial state to a specified final state. If we think that the collaborative ensemble can exploit more than one collaboration pattern (i.e. combinations of roles, interactions and behaviours) for solving the same specified task, it is then reasonable to think that each chosen collaboration pattern (as shared collaborative effort) may lead to different outcomes, depending on the current (perceived) external conditions as sensed in the surrounding environment. Self-Expression is therefore the ability of an ensemble of autonomic components inserted in an open and non-deterministic environment to change its collaboration pattern during the run-time execution of a task.

As it is trivial to understand, Self-Expression is a property that naturally fits in the domain of *ensembles* of autonomic components. An ensemble is a large set of potentially heterogeneous cooperative components: heterogeneities may be found both a the level of hardware of the component (different physical devices) and also at the level of software (different behaviours, experiences, desires, ...). The first challenge we have addressed with this regard is how to characterize the design process on the point of view of the ensemble and the surrounding environment. This is a problem that can be seen under two perspectives. First, the designer of the system can think about engineering the system starting from a task, so to have an ensemble able to solve a specific problem. Self-Expression here is seen as the ability of the system to assign roles and behaviours (to the components of the ensemble) in a dynamic way, thus taking decisions according to the external conditions. The main drawback here is that we are tailoring the ensemble for solving a specific problem, and in this way the designer does not exploit the extendibility and heterogeneity features that the ensemble is supposed to show. This is because the adaptive features are related to a single specific problem. Second, which we believe is a more effective way to exploit ensembles, one can design systems by starting with an already existing (but extensible) collection of autonomic components. In this perspective, the ensemble as a whole is therefore potentially able to complete a multitude of tasks and for each of these task, each component *knows* different collaborative ways to address the specified task. Each of these ways implies different role assignments, a different topology of interaction and obviously different behaviours (as dynamically activated processes). Once a request for completing a task is received, the components will autonomously decide the fittest coordination pattern (as re-configuration of roles, interactions and behaviours); this decision obviously depends on the situation of the surrounding environment and how the designer of the system is planning to evaluate the performance of the whole ensemble. Advantages of using this perspective are represented by the fact that tasks can be dynamically scheduled and the system is able to reconfigure itself not only if functional requirement are changing, but also in case of variations in non functional requirements. Moreover, the ensemble can share code and previous experiences related to more tasks.

On the other side, this latter perspective also presents more challenges during the design phase; however, the goal of the system engineer is to model ensembles able to deal with open and non-deterministic situations in which tasks and their implementations are continuously added to an ever-growing system (second approach). Challenges are represented by finding appropriate models and abstractions for representing tasks, their implementations and for the dynamics behind run-time changes in collaboration patterns. The holonic paradigm for designing hierarchical Multi-Agent systems represents an interesting candidate for modelling ensembles able to deploy Self-Expression: the purpose of our work is to show the mechanisms behind Self-Expression by exploiting the holonic Multi-Agent System related literature.

In the following of this section we describe how holons can be exploited to model ensembles and to introduce Self-Expression (Subsection 2.1) and the corresponding mechanism to enable the change of pattern (Subsection 2.2). Before concluding (Subsection 2.4), we show a concrete example of our approach (Subsection 2.3).

2.1 HMAS and Ensembles

In this section we briefly describe the main features regarding the ensembles, and how we are going to model them. These modelling aspects are needed in order to correctly find correspondences between ensembles (and autonomic related literature) and the concept of holons.

2.1.1 Ensemble's features

When considering ensembles, the designer has to rely on abstractions and models that should enable the dynamic formation of *groups*. A group is a sub-set of components of the whole ensemble. According to the environment dynamics, these groups can be created, merged, erased, ... We can even think at dynamically designate groups or whole ensembles according to requirements or characteristics that vary over time: this strongly relates to the concept of having a dynamically designated ensembles as target of inter-component communications (as presented in the Software Component Ensemble Language, SCEL, in [DNLPT14]). *To designate* an ensemble means to subdivide the whole set of components in groups so to expect different behaviours and/or interactions among different groups. *To dynamically designate* an ensemble implies that the criteria in which this subdivision occurs can vary over time in an unpredictable fashion.

The designer can therefore think about assigning tasks to specified groups or vice-versa: components of the group could decide to start operating on a specific task. This latter aspect relates to Agent Based Modelling [Axe97], since components are seen as entities able to deploy autonomous behaviours.

From an abstract point of view, groups of components should be addressed as single entities. These representative entities will then expose all the information regarding what they can achieve (in term of tasks) and how are they going to operate (in terms of different collaborative efforts for solving the same task, i.e. collaboration patterns).

We can think about modelling the task as bringing the state of the observed system (as sum of all of its observable variables) from a specific set of *initial* states to a state belonging to a set of *final* states. By doing so, we can think that the ensemble can choose among different collaborative efforts, and each of this effort corresponds to different collaboration patterns. According to the currently perceived condition of the surrounding environment, each choice of collaboration pattern results in a different quality on how the task is obtained; sometime a wrong choice can even prevent the ensemble to reach its goal at all. The SOTA methodology [ABZ12b] provides useful insights for modelling states, functional and non-functional requirements for these kind of modelling choices, that are necessary in order to study Self-Expression.

Self-Expression is therefore assigning the *fittest* collaboration pattern to the respective groups that compose the ensemble, and this assignment should follow the dynamics of the changes of the surrounding environment.

Encapsulation is also an important requirement when dealing with Self-Expression: external observers should know which task is currently under completion by the group, without being aware of all the details of the chosen collaboration pattern.

The Holonic Multi Agent System (HMAS) paradigm [RGH$^+$07] serves as a base of inspiration regarding these aspects and in the following section we will briefly introduce their fundamental aspects to then motivate how they can be exploited for the design process of autonomous ensembles.

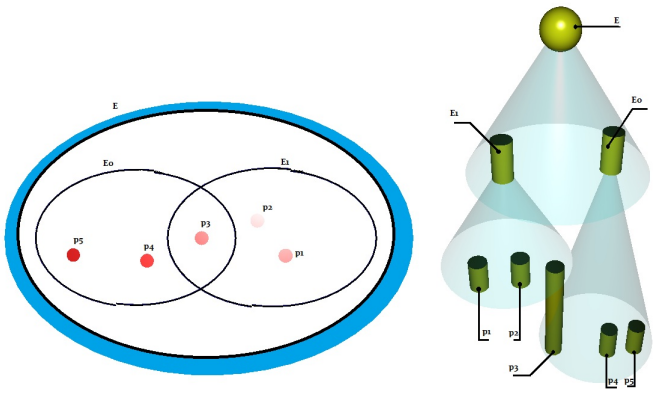

Figure 1: (left) A generic ensemble E and two sub-set of it (E_0 and E_1), with their components p. (right) Its holarchy transposition

2.1.2 HMASs for Self-Expression

A holon is a single entity and a composition of sub-entities at the same time. In other words, it is a self-similar structure since a holon may be composed of other holons, so that a moderator or head entity acts as a representative of its parts: a group (as subset of a dynamically designated ensemble) is a single entity but yet a composition of a multitude of service components, in the same way that a holon is a composition of its parts.

In Figure 1-left, a generic ensemble E can be thought as composition of two groups. These latter parts are then characterized by the presence of multiple components and since ensembles can be dynamically *designated*, the component p_3 is common to both the sub-ensembles (i.e: satisfies both the requirements for creating E_1 and E_0). The next Figure 1-right is the holonic representation of Figure 1-left by means of a holarchy. Roles and interactions are not indicated, however they are important when defining *capacities*. *Capacities* are specific interfaces presented by the holons. A capacity is defined as a description of a know-how/service, in other words, it is basically a formal description of an interface in which the developer has to specify that the task we want the ensemble to carry on:

- is identified by an ID;
- has a specific input state or a set of initial states;
- has a goal to achieve as a final state or a set of final states;
- desires to met non-functional requirements (quality or utilities).

In addition to that, we can think about adding another attribute for defining a task, that relates to the concept of *priority*: in an open and non-deterministic environment, a cooperative ensemble that is currently operating on a specific task, could decide to temporarily suspend the execution of the current task and to start executing another one, if this latter one has a higher priority index. For example: A generic ensemble of robots that is currently exploring a supposedly unsafe area, could decide to stop the exploration in case one of the component senses the presence of a fainted human being in the area. This contingency triggers a task with a higher priority index, that is represented by dragging the human being outside the unsafe area.

Each capacity can relate to different *implementations* as different ways for completing the task as described by that capacity.

The example in [RGH+07] refers to the use of different algorithms (Dijkstra, Bellman-Ford and Ant Colony Optimization (ACO)) for solving the task of finding the shortest path in a weighted graph. The authors later propose a holarchy (as hierarchy of holons) in which every organization level presents holons able to implement the task *find shortest path* with a specific (and different from level to level) implementation; moreover, in each level, roles and interactions among holons are highlighted.

Both the concepts of capacity and its different implementations can come handy when modelling Self-Expression: we can think that a collaboration pattern is a capacity implementation since in a holonic point of view, for each of these implementations corresponds an organizational level in which behaviours, roles (as specific statuses inside the same organization) and interactions (how parts in the same level influences each other) characterize a set of holons. In our case, holons are components (and at the same time, subsets of components) in dynamically designated ensembles.

Dealing with a single entity as representative of a group (as dynamically designated subset of ensemble) allows the designer to deal with all the subcategories of capacities defined in HMAS literature; specifically in our case, a task and its suitable collaboration patterns can be *atomic* (if exposed by a single component), *liaised* (if exposed by summing capacities belonging to a subset of the ensemble) and *emergent* (the case in which capacities result from interactions among components).

In the following section, more details will be provided on how tasks and their implementations can be managed for deploying Self-Expression.

2.2 Mechanism to Enable Self-Expression

Every component of the ensemble has an interface that on the one side makes public all the attributes and necessary information, while on the other side, hides the complexity of its internal structure. This latter one is composed of three sets.

- A knowledge repository;
- A set of known collaboration patterns;
- A set of known tasks' representations.

In section 2.1.2 we stressed that the description of a task (capacity) can be represented by several attributes that we can put together so to create a tuple like this:

TASK = $\langle$ id, input, output, expected quality, priority index $\rangle$

Now let us suppose to have an ensemble of potentially heterogeneous components that is supposed to be able to solve many tasks and each task can be addressed with more than one collaboration pattern. We will describe this mechanism for Self-Adaptation by explaining all the necessary steps involved.

Step 1:

An order coming from outside the system (or a particular contingency), imposes that the ensemble (or a subset of it) addresses a specific task (called T in the following).

We can dynamically designate a subset of the ensemble by communicating to all the components, the need to solve T. In this way, we are making a first split of our ensemble so to create at least two groups, i.e discriminating the subset of the ensemble able to perform T. Being able to perform T means that the component should satisfy some requirements. The nature of these requirements is specific to the case study, however, in order to clarify this we can consider components that are

not currently busy with other tasks (the priority index value should help avoiding problems relating this scheduling). Other requirements, for the sake of the example, could be related to their physical location (if applicable) or their actual resource consumption measure, and so on.

After this selection has been accomplished, the first level of our holarchy is created; by considering all the components satisfying the aforementioned requirements as a single holonic entity (thus representing the designated subset of the ensemble), we can ideally start to think about this newly designated group as a unique component.

From now on we will refer to this new component as H. H is therefore the representative of all this newly designated group: a group as container of all the candidates able to solve T. H will be the only occupant of the first level of the holarchy that we are now constructing.

Step 2:

H has now to ask itself for obtaining a list of possible implementations for T. In other words, it has to explore all the possible alternatives (in terms of different collaboration patterns) that should be feasible for solving the proposed computational problem. All the elements of the group represented by the holon H that own at least one implementation of T will communicate to H a tuple containing a string identifier, a specific expected quality for solving the task with this implementation, and a final attribute as a list of conditions (as external environmental features) for which the ensemble using that capacity implementation can guarantee the expected quality. We call these tuples as collaboration pattern tuples[2]. To *own* an implementation (i.e. a collaboration pattern) means having the set of processes (as dynamically activated local computations) able to define roles, interaction protocols among components and behaviours that will bring the system from a initial state to a final state as specified in the capacity description of T.

Step 3:

At this point, certain elements of the group (the ones who are able to receive the communication of the previous step) have to answer the enquiry about what kind (and how many) capacity implementations they own. So let us suppose to create as many groups as the number of total different implementation for T and to fill these sets with all the components of the ensemble that own all the processes related the specific implementation. Trivially, a single component can own more then one implementation for solving the same task. A second level of the holarchy is therefore now created and it is composed by one holon per existing implementation. Let us suppose that the possible implementations for solving T are name I_0 and I_1, two holons will take their place on the second level of the holarchy. A lower (third) level is created to then allocate the *single* components that belongs to the sub-sets of these newly created holons. This generic example situation is depicted in Figure 2. If a single component own more than one implementations, it will occupy more than one sub-levels in the third and level of the holarchy.

In figure 2 an example situation shows how H is created after step 1, while the other two holonic organizational levels are created at the end of step 3. In order to underline how different implementations can relate to different collaboration patterns, sample interactions among components are marked in red lines. I_0 underlines a peer-to-peer topology in which communications (or other kinds of interactions) can take place indifferently among components, while in I_1 a master component interacts with two slaves, with these latter ones are not able to communicate between themselves.

Step 4:

In this step, H is now able to construct a table (e.g. table 1). In each row of this table we can find every tuple containing a description of the collaboration patterns defined in steps 2 and 3, so to have H

[2]By simulating the single addressed task with single coordination patterns without any other added feature, the components are able to know the values that compose a single collaboration pattern tuple. Later (after deploying the whole system) they will be able to update/add/remove those values according to their experience

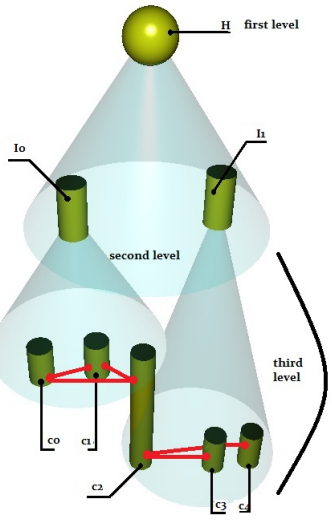

Figure 2: Holarchy transposition during step 3. Each implementation creates different holons, in this case two different implementations (I_0 and I_1). A third level underlines which component c owns processes for the different implementations. In this example the component c_2 owns processes regarding both the implementations.

able to select the fittest collaboration pattern according to both environmental conditions and desired non-functional requirements to achieve. All this tuple are related to the same specified T.

From a modelling point of view this table is stored in the knowledge repository of H; however, H is a collective component (a holon) therefore in a real world application, we can think about this table as stored in the knowledge repository of a single elected component or otherwise, in every knowledge repository of all the components of the designated group.

Table 1: Example table of collaboration pattern tuples

K	*Implementation*	*Quality*
k_0	I_0	Exp U_t'
⋮	⋮	⋮
k_1	I_x	Exp U_t''
k_2	I_x	Exp U_t'
k_3	I_y	Exp U_t'''

Each row of table 1 is defined by values of tuples K. These attributes represent all the important features regarding the surrounding environment in which the ensemble is located.

Implementation is the actual collaboration pattern chosen by the whole ensemble, identified by an id (as different implementation of T). The final column relates to the expected qualities (as non-functional requirements).

To sum it up, the mechanisms regarding the interactions among the components is here seen as which collaboration pattern (as T implementation) is supposed to be adopted if conditions k are satisfied, so to solve the task with quality Exp U. As stressed in step 3, each holon representing all the components owning a capacity implementation will communicate their row of the table to the main holon H.

Step 5:

Once the holarchy (and the related table) is complete, the ensemble will select the row of table that has the closest external conditions K (as sensed from the environment) and the closest expected utilities (as defined in the initial description of T). The corresponding implementation will lead to the fittest collaboration pattern. In case of external conditions changes or even the non-functional requirements expressed in the T tuple should experience changes, this row selection has to be done again.

2.3 Illustrative Example in Swarm Robotics

In this section we briefly apply the proposed approach in a specified example task. The task is represented by having an ensemble of robots initially randomly distributed in a confined space called *arena*. The robots have to distribute within the arena and start exploring it. The capacity of this task is represented as follows (as for now we don't consider the role of the qualities and priority indexes):

- id: exploreArea;
- Input: Ensemble randomly distributed in an unexplored arena;
- Output: Explored Arena.

Considering implementation we can identify three main collaboration patterns for addressing the task:

- Swarm approach (id: SW) All the robots randomly diffuse in the area and mark sub-areas with digital pheromones. If a robot detects a pheromone, a repulsion effect will take place, causing this latter robot to move (and therefore to explore) other sub-areas;
- Master Slave (id: MS) A robot provides orders regarding sub-areas to explore to a set of slaves;
- Peer-to-Peer (id: p2p) Robots will ideally subdivide the area to then negotiate sub-areas to explore.

Step 1:
An external command or a contingency enforces to solve
Texplore = ⟨ exploreArea, input, output ⟩
So every component from the ensemble that are located inside the arena will, from now on, be addressed as a single entity called *HA*.
Step 2:
HA interrogates all its constituents components on which implementation of Texplore they own.
Step 3:
The following communications (containing collaboration pattern tuples) take place:
FROM: all the components owning capacity impl. SW:
⟨ SW, k', expUt'⟩
TO *HA*;
FROM: all the components owning capacity impl. MS:
⟨ MS, k", expUt"⟩
TO *HA*;
FROM: all the components owning capacity impl. p2p:
⟨ p2p, k"', expUt"'⟩
TO *HA*.

k': refers to all the conditions in which it would be suitable to have a swarm approach in an area exploration task, such as a sufficiently large number of units available.
k": refers to the condition for optimally exploiting a master slave approach, such as presence of robots with additional sensing capabilities.
k"': refers to the conditions in which a p2p approach is possible, like communication possibilities and easiness to identify sub-areas before negotiation.

Depending on how these three patterns are implemented, we can think that a swarm approach will perform better in terms of minimizing exploration time, while a p2p negotiation could equally distribute energy consumption among the components. The master slave approach could both minimize time to competition and equally distribute the workload, however, MS is the less robust of the three (single point of failure).

Step 4:
HA constructs policy table 2, according to the depicted holarchy.

Table 2: exploreArea table of defined collaboration pattern tuples

K	$Implementation$	$Quality$
k'	SW	Exp U_t'
k''	MS	Exp U_t''
k'''	$p2p$	Exp U_t'''

Step 5:
HA selects (and thus enforces for all the ensemble) the policy whose k values are closest to the actual values sensed from the environment. Dynamic changes to these latter features will lead to dynamic assignments of different implementations.

2.4 Conclusion and Future Work

The holonic paradigm allows the designer to engineer ensembles with self-expression capability, making them potentially able to solve a multitude of tasks, thus to exploit heterogeneity at the level of code. As an additional very important considerations, the modeling with holons, while facilitating design and development, also support the possibility of programming self-expression patterns with the SCEL language [DNLPT14], as we have extensively described in [CCC+14].

As areas for future work, it could be really interesting to investigate how the components can change the inner parameters of the coordination patterns they could chose for completing a task. Evolutionary computing is a strong candidate for proposing methodologies for the ensemble to *evolve*, thus autonomously producing additional implementations.

3 Engineering Emergence: an Approach based on Problem Decomposition

Complex systems are made of simple elements and are characterised by the presence of non-linear interactions between them, no central control and the appearance of emergent behaviours at the system level [Mit09]. In particular, emergence is the appearance of high-level behaviours resulting from low-level simpler rules [DWH05] and an important mechanism governing these systems is self-organisation: autonomous change of the elements organisation without external control [DWH05].

Multi-Agent Systems (MAS) is one field where self-organisation and emergence are studied and applied to engineer self-adaptive systems [DMSGK06]. Some aspects of the global functionality are not explicitly pre-designed but emerge at runtime through this self-organising process in an endogenous and bottom-up way: the agents are unaware of the organisation as a whole [PHBG09]. Here, we assume self-organisation as the principle followed to design the low-level rules that lead to emergence.

Thus, the general challenge is engineering self-adaptive self-organising complex systems that exist in and modify a complex context while meeting complex requirements, in the continuation of [CPZ11, ABZ12a].Towards that goal, we propose to look at practical methodological guidelines to accompany their design. In the following, we use the term *Self-Organising MAS* (SOMAS) to denote such engineered system.

The contribution of our approach is proposing and rationalising the following design strategy: when designing SOMAS, it is necessary to follow the problem organisation. It means mapping the SOMAS decomposition on the problem decomposition in elements (and not sub-problems), and relying only on the problem abstractions for the behaviours. This strategy is not a method by itself but a complement to existing approaches and methods. It is illustrated with a running example: a swarm of bots exploring and rescuing victims in an unknown environment. Everything presented is fully implemented (See `http://www.irit.fr/~Victor.Noel/unimore-ascens-idc-2014/`).

3.1 Following the Problem Organisation

3.1.1 Problems, Requirements and Design Constraints

A problem to answer is made of a context and requirements: engineering is finding a software solution, here the SOMAS, satisfying the requirements in that context [HRJ08].

As an example, in a robotics scenario, we look at the search and secure problem: bots must explore an unknown environment to look for victims and then secure them (supposedly to rescue them, but this is not covered in our example). The context is composed of the bots (controlled by the software to build) with limited communication capabilities, the environment that have walls, the victims that must each be secured by several bots. The requirements are to search and secure victims, to secure them all, as fast as possible, to explore the accessible space fairly, to completely explore the space in a non-random way, etc.

It is important to highlight the distinction between the problem answered by the choice of using self-organisation and emergence, and the design constraints it implies: existing works characterising self-organisation and emergence do not usually explicit this distinction [DWH05, DMSGK06, PHBG09, HG03].

The following can be part of the problem: to have self-adaptation, a distributed deployment context, large-scale system, non-existence of an efficient centralised solution or impossibility of expressing the global behaviour of the system [ANM14]. Inversely, the following are mandatory design constraints when engineering SOMAS: the fact that the decision must be distributed and decentralised, that the global macro-level behaviour and organisation can't be predefined or that self-organisation must be a bottom-up process initiated locally by the elements of the system [HG03].

3.1.2 The Strategy

When designing SOMAS, two main activities are of importance: decomposing the solution in agents and giving them a self-organising behaviour. It is usual in software engineering to first model the problem space before entering the solution space [HRJ08]. However, here, we closely map the solution decomposition in agents on the problem organisation, and only rely on the problem abstractions to design their behaviours.

By problem organisation, we mean the identification of the various elements participating in it and of the role they play with respect to the requirements. We call it the *organisation* to avoid confusion with the meaning usually associated to a decomposition of the problem in sub-problems.

In the robotic example, the elements participating in the problem are the bots, but also the victims and the environment. Then, in relation with the requirements, the elements play the following role in the problem: bots moves to directions they choose, bots communicate with other bots, bots perceive victim, bots perceive walls, victims are situated, victims need a specific number of bots to be secured, etc. Inversely, an example of a potential decomposition in sub-problems would describe the problem as being about exploring and discovering on one hand, and securing collectively on the other hand.

The problem organisation can be imposed by the context (that the engineer can't control or modify) or must be chosen by the engineer when building the system. How to do so is an open question that we don't answer here.

Based on this modelling of the problem, we map elements of the solution (software agents) to the elements of the problem, and we give them the same capabilities as in the problem domain and not more. Their behaviour should be designed locally with respect to the relations elements have in the problem. The decisions (including those of their self-organising behaviour) they take should only rely on the problem domain abstractions and no higher-level global abstractions should be introduced.

In the robotic example, bots must choose where is the best direction to go at every given moment. For that, they can use what they directly see (victims and explorable areas), and when they don't know what to choose, they should rely on information shared by other bots about the state of the world with respect to the problem: where they are needed for victims or exploration. Hence, bots that see victims or explorable areas advertise about it. This information can be propagated by the bots and they can use it to decide where to go next.

Of course the complexity of the context and of the requirements (e.g., high number of bots, unknown scattering of the victims or limited perception means) are likely to make all these choices difficult. Correctly choosing the best action to take is thus an important questions: we don't pretend to answer it here, but, as said before, we argue that such decisions must rely on the problem domain abstractions. Still, we comment on this question in the next section.

3.1.3 Relation to the Design of Self-Organisation

The strategy presented in the previous section can thus be used to design a SOMAS, but, as we highlighted it, is not enough by itself. In particular, a very important point is the problem of taking the correct local decision for the agents. Some approaches to self-organisation propose local criteria to be followed by the agents in order to drive the self-organisation. For example, the AMAS theory [GGC11] is such an approach. Its main design strategy is that agents must have a cooperative social attitude: the whole approach rests on the theory that if the agents of a system are cooperative with the system environment as well as internally, then the system will behave adequately with respect to the objectives of the agents and of their environment. By identifying local non-cooperative situations agents can face, the engineer designs the agents so that they prevent or correct such situations in order to put the system in a state as cooperative as possible. Usually, a measure called criticality that is shared amongst agents is used to reflect the importance of some state of the problem and to give an agent a way to decide between several choices.

In the robotic example, a bot often has the choice between several directions and do not see any victims. In order to take the most cooperative decision, he needs some information about the state of the system: bots can advertise for example about the direction they chose to go to and some measure (the criticality) of how much more bots are needed in this direction. When a bot propagates this information (because he chose the direction), it will update this criticality in order to reflect his

and others participations in the self-organisation process: its choice means that this direction is a bit less critical now. Because bots assume they are all cooperative, they know that a direction chosen by another bot is presently the most important one to go to: choosing the most critical direction amongst all the neighbouring advertisements is enough for a bot to decide where to go next. Every decision taken will then influence how the bot computes this criticality, and inversely.

The way the self-organising process can be designed with this approach heavily relies on the fact that the agents does not contain any pre-defined behaviour in relation to the expected global behaviour, but only concepts manipulated in the definition of the problem itself, which serves to base the local decision on. For example, the criticality measure used in the AMAS approach reflects some aspect of the problem state in a comparable form: no extra high-level characterisation of what is or not a good global solution is used.

3.1.4 Rationale

The rationale behind the defended strategy, namely to follow the problem organisation when designing a SOMAS, relies on the design constraints highlighted in Section 3.1.1 and can be discussed in two cases: why follow the problem for decomposition and why use the problem organisation as we defined it here.

If the the engineer of a SOMAS introduces extra concepts foreign to the problem organisation, this means that when facing local decisions, the agents must translate their interpretation of the current state of the problem to the extra abstractions. Going far from the problem implies that we pre-set how situations are interpreted by the agents: it prevents them from interpreting correctly unforeseen situations because the concepts they manipulate can't capture them. In other words, the farther the design is from the problem, the lesser adaptive the system will be, and the lesser adequate behaviours can emerge.

In the robotic example, if the bots are designed so that to explore, they move in the direction of a repulsion vector from other bots (a typical algorithm for bots dispersion), then the problem solved is not about exploring while securing anymore, it is about dispersing bots in an environment: for example, in a hallway, a stopped bot securing a victim will prevent other bots to go behind him. Inversely, if bots behave as explained before, when the collective would profit from dispersing, then bots disperse as a result of going in directions advertised by others where the less bots are going and when there is only one direction to go (e.g., a hallway), bots just go there because it is the only advertised direction.

Then, a problem decomposition in sub-problems calls for solving each sub-problem separately (if it is not the case, then the decomposition in sub-problems is useless for the design and this is out of the scope of the discussion here). This means that the sub-solutions must then be integrated together in the agents, and such integration is embedding the complexity of the problem.

In the robotic example, if the bots have a behaviour to explore and discover victims, and another to go help other bots secure their discovered victims, it becomes very difficult to handle at the agent level the choice between going in a direction or not: it could be needed to secure a victim, but there may be already other bots going there, so it must gather information about that, and then it could not be needed because other bots are going, but then maybe there is more to explore behind the victim, so it should go anyway, except in the case where there is still enough bots going there for the same reason as it is, and so on...

This puts back the complexity of solving the problem at the agent level instead of making it the result of the collective behaviour: it is the very reason why the paradigm shift proposed by self-organisation and emergence engineering was proposed in the first place. This matter has been discussed many times in the literature (the *complexity bottleneck* [HG03]): we don't bring new arguments

for it.

3.2 Related Works and Discussion

The question of engineering emergence has been studied in various contexts, we discuss some of them and show how they are different from our contribution. On the whole, there is three ways of engineering emergences: ad-hoc, reusing or following a well-defined methodological approached.

First, ad-hoc means there is no explicit rationale behind decisions taken during the design: this is clearly out of scope of the current discussion as we are interested in ways to improve the engineering of SOMAS.

Then, reusing is usually done through the reuse of existing self-organising mechanisms that are well-known and understood. The main example of that is nature-inspired self-organisation [DMSKRZ04]. These works rely on approaches or mechanisms dependent to a certain class of problems: they are easier to apply and to reuse when possible, but in exchange it is needed to translate the concepts manipulated in the problem to the abstractions of the solution reused. It is on that point that it diverges from our work: this means that part of the original problem is lost during that translation, and we precisely advocate for relying extensively on the problem.

Finally, methodological approaches are the closest to our work in terms of motivation. We cited in Section 3.1.3 the AMAS approaches and similar strategies are for example well discussed in [DMSGK11]. All these works mainly proposes way to design the self-organising behaviour of the agents but mostly don't discuss (or rationalise) the decomposition in agents of SOMAS: this is what we do here. Other works note that simulation can be used to accompany the engineering of emergence in order to iteratively change the design with respect to the observed results: it is called *co-development* [ASW10] or *disciplined exploration* [PEC09]. Our contribution is well coherent with these approaches, but of a different nature, and show that it is possible to exploit the problem organisation to reduce the development effort of SOMAS.

3.3 Conclusion

The specificities of self-organisation and emergence rationalise the need for decomposing the system by following the problem organisation. Of course, such an absolute assertion must be instantiated differently depending on the actual problem to solve: at the very least, this strategy and rationale improve the understanding of the relation between the problem and the solution decomposition. There exist many more other issues to explore on this subject, like how to well model the problem organisation, and other related subjects, like how the problem and the solution decomposition impacts the decentralised decision making.

4 Extension to the Adaptive Architecture for Adaptive Self-Awareness

In D4.4 as well as in [BFZ14a], we have described and adaptive architecture for context-awareness.

The architecture is structured around three layers, namely *sensor*, *classifier* and *awareness* layer. Each layer can host multiple modules connected each other via application-definable topologies. The data flow from sensors (i.e., both hardware and software) through the whole architecture by means of in-memory queues enabling modules decoupling and many-to-many asynchronous communications. Each layer can host multiple modules (i.e, sensors, classifiers, awareness modules, queues). The sensor layer hosts modules that are in charge of retrieving raw data from physical/social sensors and preprocess them. The classification layer hosts modules that consume data coming from the sensor layer and classify them (i.e., generate semantically richer information). The awareness layer hosts

modules consuming labels produced in the classification layer and feeding external applications with situational information.

These modules in the awareness layer might have different goals depending on the application. However, they could be divided into two main classes. The former comprises modules delegated to sensor fusion processes. These modules receive labels, eventually conflicting, coming from multiple classification modules and apply algorithms to achieve higher semantical levels. The latter, instead, is related with the capability of the framework of monitoring and controlling itself. In a sense, the awareness layer could be the key of building a *self-aware* awareness module. For example, it would be possible to integrate within this level modules observing the internal status of the framework and activating different classifiers and sensors depending on operating conditions. This capability could be used to achieve both improved classification accuracies and reduced power consumption levels by continuously selecting to most suitable classifiers and sensors. The strategies used to drive reconfiguration might vary depending on the application.

As part of the work of the fourth year, we have extended the architecture to integrated further sensors and classifier, which will be described in Subsection 4.1, and have integrated within the framework the reasoning capabilities of KnowLang, as described in 4.2.

4.1 Improving Situation Recognition via Satellite Imagery and Commonsense Knowledge

In this Subsection, we we tackle the problem of enabling situation-recognition capabilities by fusing different sensor contributions. Specifically, we propose to extract well-known correlations among different facets of everyday life from a commonsense knowledge base. The approach is general and can be applied to a number of cases involving commonsense for the sake of: *(i)* ranking classification labels produced by different classifiers on a commonsense basis (e.g., the action classifier detects that the user is running with an high confidence and the place classifier outputs two possible labels: "park" and "swimming pool". In this case, using commonsense, it is possible to infer that the user is more likely to be in a park that in a swimming pool); *(ii)* predicting missing labels (e.g., if a user is running but the location data is missing, it is possible to propose "park" as a likely location). More in details, this subsection three main contributions: *(i)* it describes a greedy search algorithm to measure the semantic proximity of two concepts within the ConceptNet [LS04] network; *(ii)* it proposes a novel technique to extract contextual localisation data from satellite imagery; and *(iii)* it shows how to improve activity recognition accuracy by making use of two different localisation sensors and common sense knowledge.

The proposed approach is based on the assumption that commonsense knowledge can be used to measure the semantic proximity among concepts. The more two concepts are proximate, the more it is likely they have been recognized within the same context [MS08]. In this section we formally introduce the approach.

4.1.1 Problem Definition

Let us consider a set of n classifiers $C_1..C_n$, each one delegated to recognize a specific facet of the environment. Each classifier C_x is able to deal with uncertainties by producing (at every time step t) m labels $l_1(C_x,t),...,l_m(C_x,t)$ for each data sample. Given that, the overall perception of the environment can be represented as a tuple $((l_1(C_1,t),...,l_m(C_1,t)),...,(l_1(C_n,t),...,l_m(C_n,t)))$.

Here, we tackle the problem of ranking all the possible tuples provided by n classifiers on a commonsense basis.

The general problem of commonsense tuple ranking can be expressed, without loss of generality, in this way: given 2 tuples both composed by commonsense concepts, $(l_1(C_1,t),l_1(C_2,t))$ and

$(l_2(C_1,t), l_2(C_2,t))$, is it possible to establish which tuple contains the most proximate concepts on a commonsense basis?

Measuring commonsense proximity requires two key conditions to be met. In particular: *(i)* a knowledge base containing both a vocabulary covering a wide scope of topics and semantic relations hard to be discovered in an automatic way; and *(ii)* an algorithm for computing semantic proximity.

The first condition is best addressed by ConceptNet. It is a semantic network designed for commonsense contextual reasoning. It was automatically built from a collection of 700,000 sentences, a corpus being a result of collaboration of some 14,000 people. It provides commonsense contextual associations not offered by any other knowledge base. ConceptNet is organised as a massive directed and labelled graph. It is made of about 300,000 nodes and 1.6 million edges, corresponding to words or phrases, and relations between them, respectively. Most nodes represent common actions or chores given as phrases (e.g., "drive a car" or "buy food"). Its structure is uneven, with a group of highly connected nodes, and "person" being the most connected, having in- degree of about 30,000 and out-degree of over 50,000. There are over 86,000 leaf nodes and approximately 25,000 root nodes. The average degree of the network is approximately 4.7.

To meet the second requirement, we started from a preliminary round of experiments with ConceptNet that led us to the following principles:

1. Proximity increases with the number of unique paths. However, this is not a reliable indicator given that even completely unrelated concepts might be connected through long paths or highly connected nodes.

2. Proximity decreases with the length of the shortest path; nodes connected directly or through some niche edges are in a short distance, hence they are proximate;

3. Connections going through highly connected nodes increase ambiguity, therefore proximity should be inversely proportional to the degrees of visited nodes;

4. ConceptNet has been created from natural-language assertions. Thus, errors are frequent and algorithms have to be noise-tolerant;

Majewski et al. recently proposed an interesting algorithm for commonsense text categorisation inspired by similar observations [MS08]. Despite having been conceived for a different problem, it can be applied to localisation as well. The algorithm is based on the assumption that proximity among concepts is proportional to the amount of some substance s that reaches the destination node v as a result of injection to node u. The procedure has been built around two key biological paradigms such as *diffusion* and *evaporation* and works as follow:

1. a given amount of substance s is injected to a node u;

2. at every node, a fraction α of the substance evaporates and leaves the node;

3. at every node, the substance diffuses into smaller flows proportional to the out degree of the node;

4. nodes never overflow. If multiple paths visit the same node, the previous amount of substance s can be incremented;

5. target nodes are ranked according to the amount of substance s received.

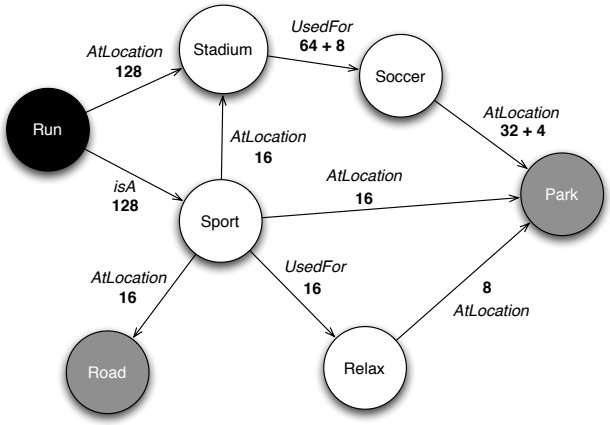

Figure 3: Concept proximity algorithm in action. 256 units of substance *s* are injected into node Run. Then, the substance diffuses over the graph and halves by evaporation ($\alpha = 0.5$) at each node it visits. The amounts of *s* that reach nodes Park and Road are 60 and 16 respectively. Park is considered more proximate than Road to Run.

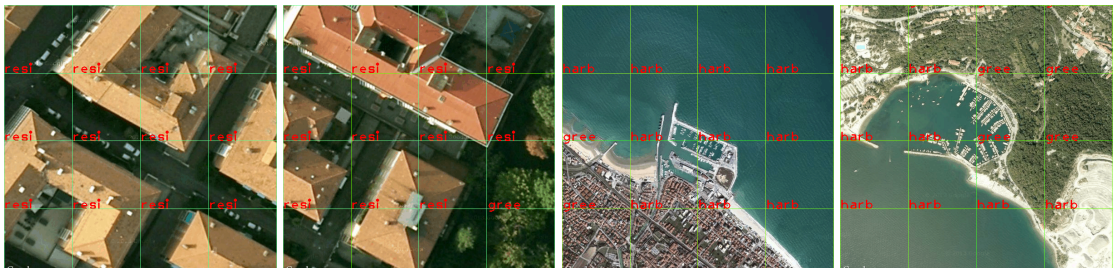

Figure 4: Four snapshot taken from the location sensor using satellite imagery. Residential and harbour areas are correctly classified. The red strings superimposed on map tiles are actual classification labels produced by the sensor. Table 4 summarizes the complete confusion matrix.

Figure 3 exemplifyies the algorithm in action. A certain amount (i.e., 256 units) of substance *s* is injected into a node (i.e., *Run*). Then, the substance diffuses over the graph and halves by evaporation at each node it visits. The amounts of *s* that reach nodes *Park* and *Road* are 60 and 16 respectively. *Park* is considered more proximate than *Road* to *Run*.

It is worth noticing that this approach can easily handle the fact that different classifiers might produce the same set of labels (i.e., classifiers observing the same facets of reality). In fact, if a label compares multiple times it is sufficient to multiply the amount of substance injected into the corresponding nodes. Furthermore, this approach permits to assign different weights to different classifiers in a straightforward way.

Finally, it is interesting to note how this algorithm matches with the principles we deduced from our preliminary studies on ConceptNet. In fact: *(i)* the evaporation process assures that short paths imply high proximity; while *(ii)* the diffusion process takes into account the total amount of connections among two concepts while diminishing the relevance of highly-connected paths.

In the following, we apply the described technique to fuse information contributions coming from sensors analyzing different facets of the same situation.

4.1.2 Improving Activity Recognition

To assess the relevance of our ideas, we used a specific instance of the general problem. We prototyped a system able to improve activity recognition accuracy by making use of two different localisation sensors. Activities are classified from accelerometer data while locations from GPS traces. All three modules have been configured to eventually produce multiple labels to deal with uncertainties. In these cases, common sense reasoning is applied.

To classify user's activities we implemented a sensor based on [BMZ10]. It collects data from 3-axis accelerometers, sampling at 10Hz, positioned in 3 body locations (i.e., wrist, hip, ankle) and classifies activities (i.e., dance, use stairs, drive, walk, run, stand still, drink) using instance-based algorithms. Furthermore, considering that human activities have a minimum duration, it aggregates classification results over a sliding window and performs majority voting on that window. Each window is associated with the most frequent label. For the sake of the experimentation, we modified it to deal with uncertainties. Instead of producing a single label for each sensor sampling, we implemented a mechanism to produce multiple labels associated with a degree of confidence. Specifically, for each sample to be classified, k nearest neighbours (associated to q classes, $k = 64$, $q <= k$) are identified. The sample is then associated to all the classes (at most 3) associated to al least $k/2q$ training samples. Table 3 reports a realistic confusion matrix for this sensor.

	Dance	Stairs	Drive	Walk	Run	Stand	Drink
Dance	0.89	0	0	0	0.11	0	0
Stairs	0	0.76	0	0.1	0.14	0	0
Drive	0.17	0	0.83	0	0	0	0
Walk	0.1	0	0	0.72	0.18	0	0
Run	0.12	0	0	0	0.88	0	0
Stand	0	0	0	0	0	0.91	0.09
Drink	0	0.1	0	0	0	0.17	0.73

Table 3: Confusion matrix of the activity recognition sensor.

4.1.3 Location Recognition via Satellite Imagery

The location sensor based on satellite images is based on GPS data and classifies user location by making use of satellite imagery. Specifically, given the GPS coordinates, it uses Google Maps API to retrieve the corresponding image tile. Then, it classifies the tile against a set of 5 categories (i.e., green, harbour, parking, rail, residential).

To implement this sensor we used an approach based on the bag-of-features [BETVG08] image classification technique. In computer vision, the bag-of-words model (BoW model) can be applied to image classification, by treating image features as words. In document classification, a bag of words is a sparse vector of occurrence counts of words; that is, a sparse histogram over the vocabulary. In computer vision, a bag of visual words is a vector of occurrence counts of a vocabulary of local image features.

A dataset comprising 200 tiles, evenly distributed among the 5 categories, has been collected from Google Maps and manually annotated. SURF features [BETVG08], chosen because of their robustness to scaling and rotation, have been extracted.

SURF features have been organised in bag-of-words representing each of the categories and used to train separate one-class SVM classifiers. During the testing stage, instead, the tile covering the user location is downloaded and tested against each classifier. The tile is assigned to the category

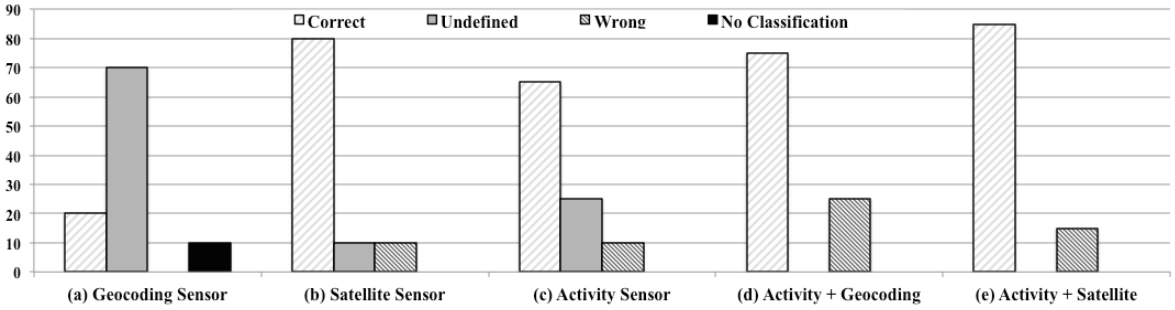

Figure 5: The reverse geocoding sensor correctly recognizes only 20% of locations (a). On the other hand, the satellite-based sensor correctly classifies around 80% (b). Finally, the activity recognition module is around 65% of correctly classified samples (c). Figures (d) and (e) show the results obtained by combing activities with locations coming from both the localisation sensors.

associated with the classifier with the highest likelihood. Table 4 shows the confusion matrix. All the classes are recognised with an accuracy comprised between 78% and 95%.

	Park	Harbour	Parking	Rail	Residential
Park	0.89	0	0.01	0	0.1
Harbour	0.07	0.86	0	0	0.07
Parking	0	0	0.78	0	0.22
Rail	0	0	0.05	0.95	0
Residential	0.06	0.03	0.05	0.05	0.81

Table 4: Confusion matrix of the localisation sensor based on satellite imagery.

4.1.4 Location Recognition via Reverse Geocoding

The location sensor based on reverse geocoding [FM11] samples GPS coordinates and classifies user's location by querying the reverse geocoding Google Maps API. Specifically, this API takes as input the GPS coordinates and a search radius and returns a list of points of interest associated to a label coming from a predefined set (i.e., road, square, park, shop, cinema, mall, restaurant, gym). Unfortunately several practical drawbacks affect this process. Google Maps database, for example, is not perfect. Although we do not have accurate statistics, we noticed that a portion of locations is still missing. Furthermore, locations' coordinates are not always precise. Finally, Google Maps does not provide information about locations' geometry. Due to this, especially for large-sized instances (e.g., parks, squares) locations can be misclassified. For example, a user running close to the border of a park is likely to be associated to the shops she is facing instead of to the park itself.

To mitigate these problems and avoid false negatives, the system has been setup to use a search radius of 250m. Clearly, the number of reverse geo-coded locations is proportional to the search radius. Because of this, especially in densely populated areas, the system might produce numerous false positives. To reduce them, while keeping an acceptable level of false negatives, we implemented 3 filters acting on the GPS signal. Specifically: the first acts on the assumption that each class is more likely to be visited during defined portions of the week. The second, acts on the assumption that each class of locations is fairly characterized by the duration of the visit. This duration is usually

related with a GPS signal interruption. Finally, the third one, filters out each label not compatible with measured speed.

4.1.5 Experimental Evaluation

To assess the feasibility of our idea, we used the system described in Subection 4.1.2 to collect a dataset comprising a full day of a single user.

The activity recognition module, has been trained to classify 8 activities (i.e., climb, use stairs, drive, walk, read, run, use computer, stand still, drink). For each class, 300 training samples have been selected. The location module implementing reverse geocoding, instead, sampled GPS coordinates each 30 seconds. GPS coordinates has been labelled with 5 different categories (i.e., street, university, bar, park and library).

We first discuss the performance of recognition modules, considered independently. Figure 5(a)(b)(c) summarizes the results. The reverse geocoding localization sensor is the less precise among the three. It correctly recognizes only 20% of locations because of multiple commercial activities are usually located within its search radius. On the other hand, the satellite-based sensor correctly classifies around 80% of the samples. Finally, the activity recognition module is around 65% of correctly classified samples.

When both location and activity labels are combined using ConceptNet, 4 cases can occur: *(i)* both are available, *(ii)* only activity is available, *(iii)* only location is available, *(iv)* no data available. The first case allows to apply commonsense sensor fusion. In both the second and the third case, instead, commonsense can be used to identify a possible place or activity to complete the *(activity, place)* tuple.

Figure 5(d)(e) show the results obtained by combing activity labels with both the location labels. A significant improvement has been achieved. It is worth noticing that the *Undefined* (i.e., multiple labels available) category is lowered to zero meaning that ConceptNet is always capable of providing a ranking of action-place couples. Furthermore, the *No Classification* data category is lowered to zero, in fact one of the advantage of the use of ConceptNet is to provide missing data. Please note that in our experiment we never experienced the concurrent lack of both sensorial data, that should have called for different strategies similar to activity and location prediction, such as bayesian networks [BCM$^+$08]. It is worth noticing that the fusion process with the satellite-based sensor produced better results because of its initial performance was batter than the reverse geocoding one.

4.1.6 Discussion

In this Subsection we have presented preliminary results we have obtained with a novel approach that combines an activity classifier and location classifier using satellite imagery with the ConceptNet knowledge base. Different classifiers are fused together on a commonsense basis for both: *(i)* improve classification accuracy and *(ii)* dealing with missing labels. The approach has been discussed through a realistic case study focused on the recognition of both locations visited and activities performed by a user. Results have been encouraging and the corresponding tools have been integrated into the self-awareness architectures.

4.2 Integration with KnowLang

As we have said the awareness layer of our self-aware architectures hosts modules consuming labels produced in the classification layer and feeding external applications with situational information. Some modules are delegated to sensor fusion processes. That is, they receive labels, eventually conflicting, coming from multiple classification modules and apply algorithms to achieve higher semantical levels. For example, as described earlier in this section, commonsense knowledge has been

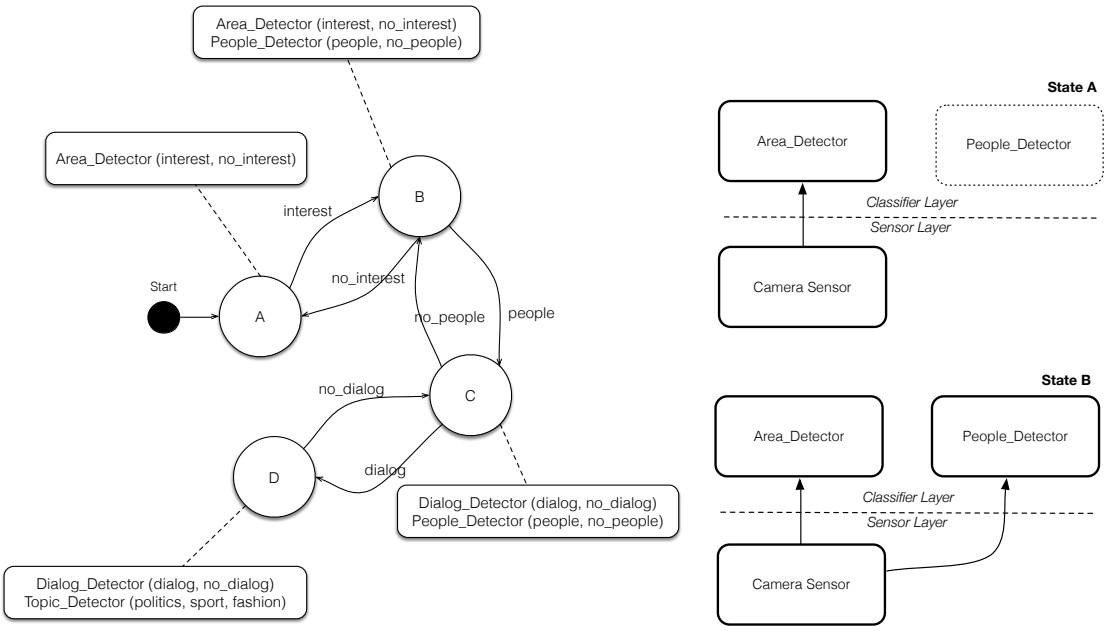

Figure 6: A self-aware surveillance drone designed for detecting people within specific areas of interest.

recently proposed and could be integrated at this level. Other modules concerns the capability of the framework of monitoring and controlling itself.

4.2.1 The Role of KnowLang

The awareness layer could be the key of building a *self-aware* awareness module. For example, it would be possible to integrate within this level modules observing the internal status of the framework and activating different classifiers and sensors depending on operating conditions. This capability could be used to achieve both improved classification accuracies and reduced power consumption levels by continuously selecting to most suitable classifiers and sensors.

In this context, we embedded within this layer the KnowLang reasoner, the formal language for knowledge representation and reasoning developed within WP3. The key role of KnowLang in the context of our self-aware architecture being that of receiving labels from the classification layer and select the coming actions.

4.2.2 Case Study

To demonstrate our approach, we describe an example of self-aware surveillance drone designed for detecting people within specific areas of interest (see Figure 6). The system has the goal to collect sensorial data, classify them and define the situation they are immersed within. However, instead of using a single complex classifier, we provide developers a way to use a number of simpler and more specific classifiers. These modules can be enabled, disabled, wired and rewired in a dynamic way by making use of their output to navigate the state automata. Each status has a name and is associated with a set of classifiers - and associated sensors - that have to be active and a set of possible transitions. Each time the output of a active classifier changes, a reconfiguration is applied. Needed modules are deployed and inactive are automatically removed to reach the new status. In this way, the overall

problem of situation recognition is modularised in a way similar we believe our brain works. Each node embeds the knowledge acquired by the former and activates more specific classifiers to collect further details.

Figure 6, drives the reconfiguration of a surveillance drone. State A is activated as soon as the drone takes off and tries to detect areas of interests. As soon as an area of interest is spotted, state B is activated and eventual people are detected. State C, activated only when people are detected in an area of interest analyses audio signals to detect dialogs. Finally, state D, refines state C by inferring the general topic of the conversation using common sense knowledge and speech recognition techniques.

It is worth noticing that this example show the internal logic of the awareness module of two different applications. However, despite the logic used to collect situational awareness have to be linked with the applicative logic, these automata are agnostic about *how* the situational knowledge is actually collected. In fact one could completely change sensors and classifiers used in each and every state without altering the applicative logic. We think this feature could both: *(i)* sensibly speed up the prototyping of pervasive applications and *(ii)* helping the development of pattern recognition modules. If fact, one could quickly asses different algorithms, libraries, approaches without altering anything within the actual application.

For the purpose of this case study, we used KnowLang to support the behaviour outlined above. The first step was to specify a simple knowledge base (KB) representing the domain outlining the case study, e.g., the drone itself and the drone's operational environment with entities such as areas of interest, people, drone base, etc. Recall that this domain is described via a domain ontology expressed through domain-relevant concepts and objects (concept instances) related through relations. To handle explicit concepts like situations, goals, and policies, we granted some of the domain concepts with explicit state expressions. The following is a partial specification of the Drone concept. As shown, the Drone has properties, functionalities, and states (Boolean expressions validating a specific state), etc.

```
CONCEPT Drone {
  PARENTS {srvllnceDrone.drones.CONCEPT_TREES.System}
  CHILDREN { }
  PROPS {
  PROP dFlyCapacity {TYPE{srvllnceDrone.drones.CONCEPT_TREES.FlyingCapacity} CARDINALITY{1}}
    PROP dPlanner {TYPE{srvllnceDrone.drones.CONCEPT_TREES.Planner} CARDINALITY{1}}
    PROP dCommunicationSys {TYPE{srvllnceDrone.drones.CONCEPT_TREES.CommunicationSys} CARDINALITY{1}}
  }
  FUNCS {
    FUNC plan {TYPE {srvllnceDrone.drones.CONCEPT_TREES.Plan}}
    FUNC lineExplore {TYPE {srvllnceDrone.drones.CONCEPT_TREES.LineExplore}}
    FUNC spiralExplore {TYPE {srvllnceDrone.drones.CONCEPT_TREES.SpiralExplore}}
    FUNC takeoff {TYPE {srvllnceDrone.drones.CONCEPT_TREES.TakeOff}}
    FUNC flyTowardsBase {TYPE {srvllnceDrone.drones.CONCEPT_TREES.FlyTowardsBase}}
    FUNC lookForPeople {TYPE {srvllnceDrone.drones.CONCEPT_TREES.LookForPeople}}
  }
  STATES {
    STATE IsUp { PERFORMED{srvllnceDrone.drones.CONCEPT_TREES.Drone.FUNCS.takeoff} }
    STATE IsPlaning { IS_PERFORMING{srvllnceDrone.drones.CONCEPT_TREES.Drone.FUNCS.plan} }
    STATE IsExploring { IS_PERFORMING{srvllnceDrone.drones.CONCEPT_TREES.Drone.FUNCS.lineExplore} OR
          IS_PERFORMING{srvllnceDrone.drones.CONCEPT_TREES.Drone.FUNCS.spiralExplore} }
    STATE InLowFlayCapacity {
          srvllnceDrone.drones.CONCEPT_TREES.Drone.PROPS.dFlyCapacity.STATES.smallFlyingTime}
    STATE FoundAreaOfInterest { srvllnceDrone.drones.CONCEPT_TREES.Drone.STATES.IsExploring AND
          srvllnceDrone.drones.CONCEPT_TREES.SpottedAreasOfInterest >= 1 }
    STATE FlayingOverAreaOfInterest {  }
    STATE IsExploringAreaOfInterest { srvllnceDrone.drones.CONCEPT_TREES.Drone.STATES.IsExploring AND
          srvllnceDrone.drones.CONCEPT_TREES.Drone.STATES.FlayingOverAreaOfInterest }
    STATE FoundPeopleOfInterest {
          srvllnceDrone.drones.CONCEPT_TREES.Drone.STATES.IsExploringAreaOfInterest AND
          srvllnceDrone.drones.CONCEPT_TREES.SpottedPeople >= 1 }
  }
}
```

To specify the drone's behaviour with KnowLang, we used goals, policies, and situations. The following is a specification sample showing a drone's policy called *GoFindAreaOfInterest*. As shown, the policy is specified to handle the goal *FindAreaOfInterest* and is triggered by the situation *DroneIsOnAndAreaNot Found*. Further, the policy triggers via its *MAPPING* sections conditionally (e.g., there is a *CONDITONS* directive that requires the drone's flying capacity be higher than the estimated time needed to get back to the base) the execution of a sequence of actions. When the con-

ditions were the same, we specified a probability distribution among the *MAPPING* sections involving same conditions (e.g., *PROBABILITY*0.6), which represents our initial belief in action choice.

```
CONCEPT_POLICY GoFindAreaOfInterest {
  CHILDREN {}
    PARENTS { srvllnceDrone.drones.CONCEPT_TREES.Policy}
    SPEC {
      POLICY_GOAL { srvllnceDrone.drones.CONCEPT_TREES.FindAreaOfInterest }
      POLICY_SITUATIONS { srvllnceDrone.drones.CONCEPT_TREES.DroneIsOnAndAreaNotFound }
      ....
      POLICY_MAPPINGS {
        MAPPING {
         CONDITIONS { srvllnceDrone.drones.CONCEPT_TREES.TimeToDroneBase <
           srvllnceDrone.drones.CONCEPT_TREES.drone.PROPS.dFlyCapacity }
         DO_ACTIONS { srvllnceDrone.drones.CONCEPT_TREES.drone.FUNCS.lineExplore }
         PROBABILITY {0.6}
      }
      MAPPING {
        CONDITIONS { srvllnceDrone.drones.CONCEPT_TREES.TimeToDroneBase <
          srvllnceDrone.drones.CONCEPT_TREES.drone.PROPS.dFlyCapacity }
        DO_ACTIONS { srvllnceDrone.drones.CONCEPT_TREES.drone.FUNCS.spiralExplore }
        PROBABILITY {0.4}
      }
      ....
    }
  }
}
```

As specified, the probability distribution gives initial designer's preference about what actions should be executed if the system ends up in running the *GoFindAreaOf Interest* policy. Note that at runtime, the KnowLang Reasoner maintains a record of all the action executions and re-computes the probability rates every time when a policy has been applied and consecutively, actions have been executed. Thus, although initially the system will execute the function *lineExplore* (it has the higher probability rate of 0.6), if that policy cannot achieve its goal with this action, then the probability distribution will be shifted in favour of the function *spiralExplore*, which might be executed next time when the system will try to apply the same policy. Therefore, probabilities are recomputed after every action execution, and thus, the behaviour changes accordingly.

As mentioned above, policies are triggered by situations. Therefore, while specifying policies handling the drone's objectives (e.g., *FindAreaOfInterest*), we need to think of important situations that may trigger those policies. Note that these situations shall eventually be outlined by scenarios providing alternative behaviours or execution paths out of that situation. The following code represents the specification of the situation *DroneIsOnAndAreaNotFound*, used for the specification of the *GoFindAreaOfInterest* policy.

```
CONCEPT_SITUATION DroneIsOnAndAreaNotFound {
....
  SPEC {
    SITUATION_STATES {srvllnceDrone.drones.CONCEPT_TREES.drone.STATES.IsUp,
      srvllnceDrone.drones.CONCEPT_TREES.drone.STATES.IsExploring}
    SITUATION_ACTIONS {srvllnceDrone.drones.CONCEPT_TREES.LineExplore,
      srvllnceDrone.drones.CONCEPT_TREES.FlyTowardsBase}
}}
```

As shown, the situation is specified with *SITATION_STATES* (e.g., the drone's states *IsUp* and *IsExploring*) and *SITUATION_ACTIONS* (e.g., *LineExplore*, *SpiralExplore*, and *FlyTowardsBase*). To consider a situation effective (i.e., the system is currently in that situation), the situation states must be respectively effective (evaluated as true). For example, the *DroneIsOnAndAreaNotFound* situation is effective if the Drone's state *IsExploring* is effective (is hold). The possible actions define what actions can be undertaken once the system falls in a particular situation.

Note that the presented specification is a part of the KB that is operated by the KnowLang Reasoner. The reasoner encapsulates that KB and acts as a module in the awareness layer of the *framework for knowledge collection*. The reasoner is "fed" with classified sensory data (labels), produced by the classification layer, and returns situational information and proposed behaviour upon request. The consumed labels help the reasoner update the KB, which results in re-evaluation of the specified concept states (recall that states are specified as a Boolean expression over the ontology, i.e., a state

expression may include any element in the KB). Consecutively, the evaluation of the specified states help the reasoner determine at runtime whether the system is in a particular situation or a particular goal has been achieved. Moreover, it can deduct an appropriate policy that may help the drone ”go out” of a particular situation.

5 Performance Awareness

Performance awareness of a software system combines multiple mechanisms – collection of performance data with high accuracy and low overhead, analysis of the collected data and detection of significant changes, as well as adaptation in response or in anticipation of such changes. In the ASCENS Deliverable D4.5, we have described the progress we have made in data collection and engineering of adaptation responses – this section of the ASCENS Deliverable D4.4 connects to the earlier contributions with robust methods of change detection.

We open with a motivating example in Section 5.1, where we explain why the deceptively simple problem of detecting performance change requires a robust solution. In Section 5.2, we define a new interpretation of the performance comparison operators of the Stochastic Performance Logic that addresses the issues illustrated by the motivating example. We briefly evaluate our approach in Section 5.3.

5.1 Motivation

Our motivating example is situated in the ASCENS Cloud Case Study. A cloud application would often exhibit decreasing performance when faced with excessive workload – and an adaptive cloud application would react to this situation by allocating more processing capacity from the cloud. We pick one such application – an example XML processing server – and look at how such reactive adaptation works.

We envision a feedback adaptation mechanism that measures the request processing time and when the time exceeds a threshold, it launches a new XML processing server instance. Testing reveals that in normal conditions, the server should exhibit an average request processing time of around 100 ms, we would therefore use this average plus some slack to set the threshold. Upon deployment, the simple adaptaion design immediately backfires – the very first request processing time we collect is over 900 ms, triggering an adaptation. Obviously, the very first request cannot represent an excessive workload and the adaptation is not correct.

An obvious explanation for the initial failure is that the initial measurement is distorted and therefore invalid. This is a common enough occurrence, we therefore collect more measurements – enough to filter out distortions but not enough to cause severe overhead. The results for the first 30 measurements are on Figure 7.

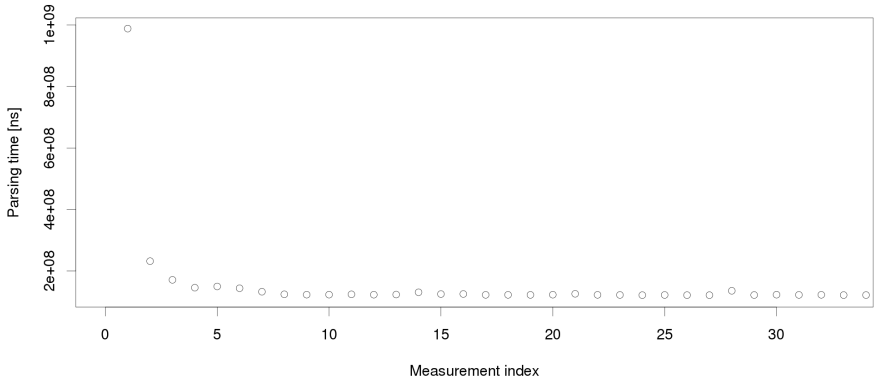

Figure 7: Server request processing time (first 30 measurements)

By looking at Figure 7, we could conclude that the measurements become stable after some five observations. However, collecting even more measurements dispels this impression, as illustrated on Figure 8. The real server performance is not stable. Instead, it exhibits multiple performance modes that change at irregular intervals, and the processing time does not stabilize in a reasonably short interval.

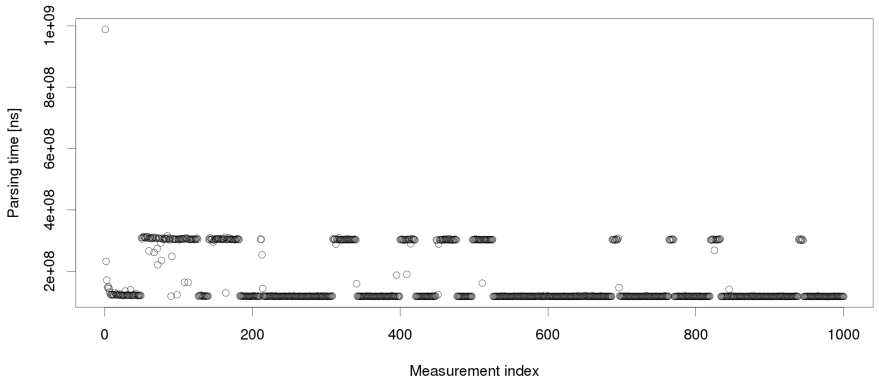

Figure 8: Server request processing time (first 1000 measurements)

To finish our example, we add more measurements after server restart, which show that the modes themselves are not necessarily stable, see Figure 9.

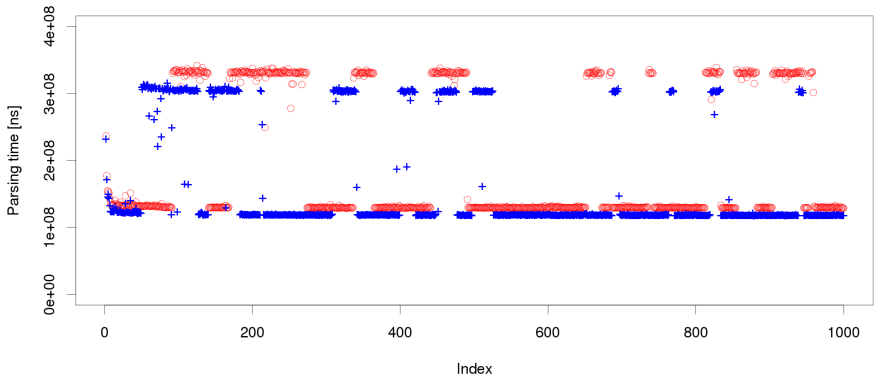

Figure 9: Server request processing time (colors distinguish different server restarts)

The lack of performance stability is not merely an artefact of our example server. Similar behavior can be observed with many software systems, where it is often made even worse by additional measurement noise (here, we have measured the server under very stable controlled conditions to demonstrate our point). The lack of stability has direct impact on the available adaptation options – simple threshold detection is not reliable and collecting more measurements is not helpful because it inevitably introduces adaptation delays.

5.2 Performance Logic Interpretation

Earlier, the ASCENS project has introduced the Stochastic Performance Logic (SPL) [BBK+12], a formalism for expressing conditions on the observed performance. SPL provides multiple interpretations – in particular, the *expected value interpretation*, used with random variables, and the *sample based interpretation*, used with measurements that meet certain statistical test requirements. Here, we extend SPL with the *bootstrap based interpretation*, which addresses issues outlined in our motivating example, namely initial transient conditions and long term fluctuations.

5.2.1 Handling Initial Transient Conditions

As an inherent property of our computing environment, each measurement run is exposed to mechanisms that may introduce transient execution time changes. Measurements performed under these conditions are typically denoted as warmup measurements, in contrast to steady state measurements.

Many common mechanisms can introduce warmup, in Java environment it is, for instance, just-in-time compilation, biased locking [Dic06, DMS10] or automatic heap size adjustment [Ora14].

Warmup measurements are not necessarily representative of steady state performance and are therefore typically avoided. Often, this can be done by configuring the relevant mechanisms appropriately (a good example is the heap size adjustment, which can be simply disabled). Unfortunately, the configuration changes that help reduce the initial transient conditions sometimes also impact the steady state performance.

Given these obstacles, we believe that warmup should not be handled at the level of logic interpretation. Instead, knowledge of the relevant mechanisms should be used to identify and discard observations collected during warmup.

5.2.2 Handling Long Term Fluctuations

To support systems whose performance exhibits long term benign fluctuations – that is, changes in performance that do not require adaptation, our bootstrap based interpretation first determines the statistical distribution of the fluctuations. Once the distribution is known, the interpretation can test whether an observed change can be a fluctuation.

In line with our experiments on existing software [HHK+13], we assume the system execution consists of *runs*, defined as periods between benign performance fluctuations. We assume that all observations $P_M^{i,j}(x_1,\ldots,x_m)$ in a run i are identically and independently distributed with a conditional distribution depending on a hidden random variable C. We denote this distribution as $B_M^{C=c}$, meaning the distribution of observations in a run conditioned by drawing some particular c from the hidden random variable C.

When testing for a significant difference between performance observed during episodes M and N, we define the distributions of the test statistics as follows:

- $B_{\overline{M},r_M,o_M}$ is the distribution function of $(r_M o_M)^{-1}\sum_{i=1}^{r_M}\sum_{j=1}^{o_M} tm(\dot{P}_M^{i,j}(x_1,\ldots,x_m))$, where $\dot{P}_M^{i,j}(x_1,\ldots,x_m)$ denotes a random variable with distribution $B_M^{C=c}$ for c drawn randomly once for each i. In other words, $B_{\overline{M},r_M,o_M}$ denotes a distribution of a mean computed from r_M runs of o_M observations each.

- $B_{\overline{M},r_M,o_M-\overline{N},r_M,o_M}$ is the distribution function of the difference $\tilde{M}-\tilde{N}$, where $\tilde{M}$ is a random variable with distribution $B_{\overline{M},r_M,o_M}$ and $\tilde{N}$ is a random variable with distribution $B_{\overline{N},r_N,o_N}$.

After adjusting the distributions $B_{\overline{M},r_M,o_M}$ and $B_{\overline{N},r_N,o_N}$ by shifting to have an equal mean, the performance comparison can be defined as:

- $P_M(x_1,\ldots,x_m) \leq_{p(tm,tn)} P_N(y_1,\ldots,y_n)$ iff

$$\overline{M}-\overline{N} \leq B^{-1}_{\overline{M},r_M,o_M-\overline{N},r_N,o_N}(1-\alpha)$$

  where $\overline{M}$ denotes the sample mean of $tm(P_M(x_1,\ldots,x_m))$, $\overline{N}$ is defined similarly, and $B^{-1}_{\overline{M},r_M,o_M-\overline{N},r_N,o_N}$ denotes the inverse of the distribution function $B_{\overline{M},r_M,o_M-\overline{N},r_N,o_N}$ (i.e. for a given quantile, it returns a value).

- $P_M(x_1,\ldots,x_m) =_{p(tm,tn)} P_N(y_1,\ldots,y_n)$ iff

$$B^{-1}_{\overline{M},r_M,M-\overline{N},r_N,o_N}(\alpha) \leq \overline{M}-\overline{N} \leq B^{-1}_{\overline{M},r_M,o_M,-\overline{N},r_N,o_N}(1-\alpha)$$

An important problem is that the distribution functions $B_{\overline{M},r_M,o_M}$, $B_{\overline{N},r_N,o_N}$, and consequently $B_{\overline{M},r_M,o_M-\overline{N},r_N,o_N}$ are unknown. To get over this problem, we approximate the B-distributions in a non-parametric way by bootstrap and Monte-Carlo simulations [She11]. This can be done either by using observations $P_M^{i,j}(x_1,\ldots,x_m)$ directly, or by approximating from observations of other methods whose performance behaves similarly between runs.

The basic idea of the bootstrap is that the distribution of a sample mean of an i-th run $\hat{\theta}_{i,o} = o^{-1}\sum_{j=1}^{o}\dot{P}_X^{i,j}(x_1,\ldots,x_m)$ of some method X is estimated by sampling its "bootstrap version" $\theta^*_{i,o} = o^{-1}\sum_{j=1}^{o}P_X^{*i,j}(x_1,\ldots,x_m)$, where samples $P_X^{*i,j}(x_1,\ldots,x_m)$ are randomly drawn with replacement from samples $P_X^{i,j}(x_1,\ldots,x_m)$. Extending this line of reasoning to the mean of run means $\overline{X}_{r,o} = r^{-1}\sum_{i=1}^{r}\hat{\theta}_{i,o}$,

we estimate the distribution of $\overline{X}_{r,o}$, i.e. $B_{\overline{X},r,o}$, by sampling its "bootstrap version" $\overline{X}^*_{r,o} = r^{-1}\sum_{i=1}^{r}\theta^{**}_{i,o}$, where $\theta^{**}_{i,o}$ is randomly drawn with replacement from $\theta^{*}_{i,o}$. We denote $B^*_{\overline{X},r,o}$ as the distribution of $\overline{X}^*_{r,o}$.

The exact computation of the distribution of the bootstrapped estimator (e.g. the mean of run means) requires traversal through all combinations of samples. This is computationally infeasible, thus Monte-Carlo simulation is typically employed to approximate the exact distribution of the estimator. In essence, the Monte-Carlo simulation randomly generates combinations of samples, evaluates the estimator on them (e.g. $\overline{X}_{r,o}$) and constructs an empirical distribution function $F_n(\cdot) \equiv \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}(X_i \leq \cdot)$ (e.g. of $B_{\overline{X}^*_{r,o}}$ in our case), where $\mathbf{1}(A)$ denotes the indicator function of a statement A.

The whole aparatus of the bootstrap and the Monte-Carlo simulation can then be used to create the bootstrapped distributions $B^*_{\overline{X},r_X,o_X}$, $B^*_{\overline{Y},r_Y,o_Y}$ and their difference $B^*_{\overline{X},r_X,o_X-\overline{Y},r_Y,o_Y}$. To obtain the desired test distribution $B_{\overline{M},r_M,o_M-\overline{N},r_N,o_N}$, we use the approximation $B^*_{\overline{X},r_X,o_X-\overline{Y},r_Y,o_Y}$, where X,Y stand for M,N or other methods whose performance behaves similarly between runs.

With the theory in place, we define the bootstrap based interpretation of the logic as follows:

Let $tm,tn : \mathbb{R} \to \mathbb{R}$ be performance observation transformation functions, P_M and P_N be method performances, $x_1,\ldots,x_m,y_1,\ldots,y_n$ be workload parameters, $\alpha \in \langle 0, 0.5\rangle$ be a fixed significance level, and let X,Y be methods (including M and N) whose performance observations are used to approximate the distributions of P_M and P_N, respectively.

For a given experiment $\mathscr{E}$, the relations $\leq_{p(tm,tn)}$ and $=_{p(tm,tn)}$ are interpreted as follows:

- $P_M(x_1,\ldots,x_m) \leq_{p(tm,tn)} P_N(y_1,\ldots,y_n)$ iff

$$\overline{M} - \overline{N} \leq B^{*-1}_{\overline{X},r_X,o_X-\overline{Y},r_Y,o_Y}(1-\alpha)$$

- $P_M(x_1,\ldots,x_m) =_{p(tm,tn)} P_N(y_1,\ldots,y_n)$ iff

$$B^{*-1}_{\overline{X},r_X,o_X-\overline{Y},r_Y,o_Y}(\alpha) \leq \overline{M} - \overline{N} \leq B^{*-1}_{\overline{X},r_X,o_X-\overline{Y},r_Y,o_Y}(1-\alpha)$$

5.3 Evaluating Change Detection

To evaluate the practical application of the bootstrap based interpretation, we perform controlled experiments with artificially introduced performance changes. Without controlled experiments, the interpretation is difficult to evaluate – measurements always exhibit some differences and deciding whether the differences are essential or incidental is often a matter of perspective.

A practical concern related to the SPL interpretations is how many false positives (situations where the interpretation reports a significant performance difference even when none truly exists) and false negatives (situations where the interpretation does not report a difference even when one does exist) can be expected. To test this concern, we construct an evaluation scenario around code that builds a DOM data structure from an XML input. We measure the code in multiple runs with multiple input sizes and use multiple SPL interpretations to decide whether the measured performance differs for various combinations of input size and run count. In detail, for interpretation i, input size s and run count r:

1. We use random sampling to select r runs with input size s into a set of measurements M_s, to represent the observations of the method performance on input size s, denoted $P(s)$.

2. We use random sampling to select r runs with base input size b into a set of measurements M_b, to represent the observations of the method performance on base input size b, denoted $P(b)$.

3. We use the interpretation i on M_s and M_b to decide whether $P(s) \geq_{p(id,id)} P(b)$ and $P(s) \leq_{p(id,id)} P(b)$ at significance $\alpha = 0.01$.

We repeat the steps enough times to estimate the probability of the individual decisions. The results are available on Figure 10, with the interpretation from Section 5.2 denoted as *Non-Parametric* and two other interpretations denoted as *Parametric* and *Welch*.[3] Each run collects $o = 20000$ observations after a warmup of 40000 observations, the input size ranges from about 5000 XML elements in a file of 282 kB to about 5600 XML elements in a file of 316 kB (base input size). In total 130 runs were used to derive the distribution $B_{\overline{M},r_M,o_M-\overline{N},r_N,o_N}$, approximated by $B^*_{\overline{X},r_X,o_X-\overline{Y},r_Y,o_Y}$ for $X = Y = M(s)$. For each input size, 10 runs were available for random sampling.

To interpret the results on Figure 10, we first look on the measurements with zero file size difference, that is, the measurements where the interpretation was asked to decide whether there is a discernible difference between performance of the same method under the same workload. Any rejection for zero file size difference constitutes an evident false positive. We see that the Welch interpretation is incorectly rejecting the null hypothesis very often regardless of the number of runs used. The Parametric interpretation is only incorrectly rejecting the null hypothesis for a small number of runs. The Non-Parametric interpretation from Section 5.2 is almost never rejecting the null hypothesis.

Next, we look on the measurements with non-zero file size difference. Here, chances are that the file size difference is also reflected in the performance – to provide a rough estimate, it takes on average 9.04 ms to execute the method for the largest input size and 7.81 ms to execute the method for the smallest input size. We therefore want the interpretation to reject the hypothesis that $P(s) \geq_{p(id,id)} P(b)$ – in other words, to identify that there actually is a performance difference ; but this is only useful when the interpretation also never rejects $P(s) \leq_{p(id,id)} P(b)$ – in other words, it should identify the performance difference reliably. We see that the Welch interpretation is very willing to identify performance differences, however, it is not really reliable until the difference is very large and the number of runs is sufficient. The Parametric interpretation is reasonably reliable except when given a small number of runs, and the sensitivity to performance differences depends on the number of runs available. The Non-Parametric interpretation from Section 5.2 is reliable even when used with just one run.

To summarize, the Welch interpretation works reasonably well only for relatively large performance differences. The more sophisticated Parametric and Non-Parametric interpretations exhibit similar sensitivity to performance differences and are more reliable – the Parametric interpretation only with more runs, the Non-Parametric interpretation even with one run. This is not counting the runs used to build the necessary distribution of the test statistics. In practical settings from Section 5.1, new runs would be measured and assessed as machine time permits – the Non-Parametric interpretation can be used with the test statistics distribution built from historical measurements to make a quick initial assessment, and the Parametric interpretation could refine the assessment after more data becomes available.

6 Conclusions

Following the completion of some key tools in year 3, the fourth year of the activities within WP4 has brought several additional interesting scientific and technological results, in line with the planned activities, and has progressed towards the integration of the results within the project. Specifically:

- We have show how it is possible to implement self-expression patterns in SCEL and accordingly to a holonic multiagent systems perspective;

[3] A publication describing the details is currently under review.

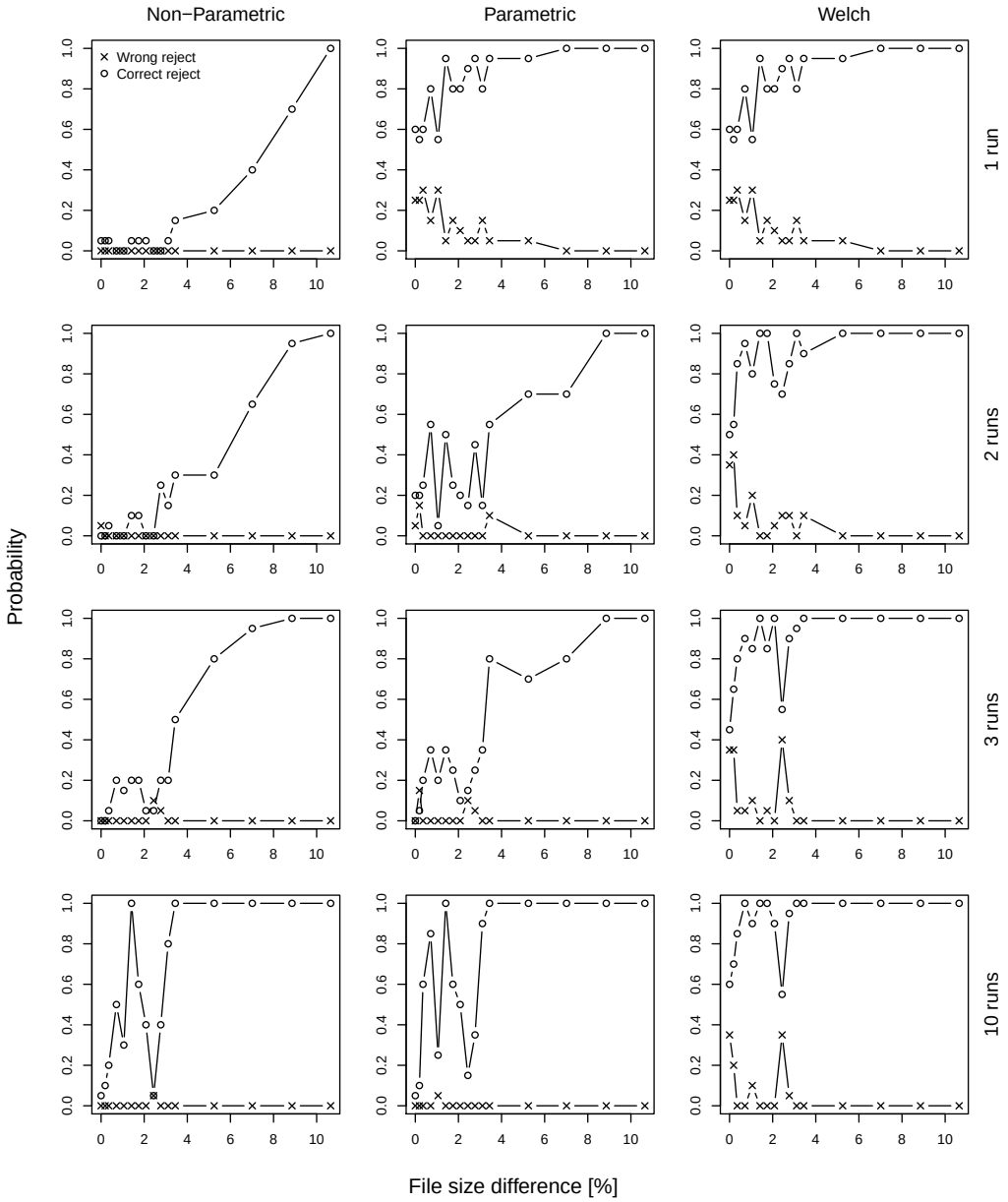

Figure 10: Sensitivity to input size change for combinations of interpretation and run count.

- We have finalized the methodological guidelines for engineering emergence;
- We have extended the features of the self-awareness architectures and integrated the KnowLang reasoning capabilities in it;
- We have progressed towards the robust definition of performance awareness tools.

In summary, WP4 has achieved scientific and technological results that are mostly in line with what expected at the beginning of the project, although with the necessary tuning of objectives to keep in line with the state of the art in the area and with the results achieved within other WPs.

References

[ABZ12a] D. B. Abeywickrama, N. Bicocchi, and F. Zambonelli. SOTA: Towards a general model for self-adaptive systems. In *WETICE Conference*, pages 48–53. IEEE, 2012.

[ABZ12b] Dhaminda B Abeywickrama, Nicola Bicocchi, and Franco Zambonelli. Sota: Towards a general model for self-adaptive systems. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on*, pages 48–53. IEEE, 2012.

[ANM14] Jean-Paul Arcangeli, Victor Nol, and Frdric Migeon. Software Architectures and Multiagent Systems. In Mourad Oussalah, editor, *Software Architectures*, volume 2, pages 171–208. Wiley, 2014.

[ASW10] Paul Andrews, Susan Stepney, and Alan Winfield. Simulation as an experimental design process for emergent systems. In *EmergeNET4 Workshop: Engineering Emergence*, 2010.

[Axe97] Robert M Axelrod. *The complexity of cooperation: Agent-based models of competition and collaboration*. Princeton University Press, 1997.

[BBK+12] Lubomir Bulej, Tomas Bures, Jaroslav Keznikl, Alena Koubkova, Andrej Podzimek, and Petr Tuma. Capturing Performance Assumptions using Stochastic Performance Logic. In *Proc. ICPE 2012*. ACM, 2012.

[BCM+08] Nicola Bicocchi, Gabriella Castelli, Marco Mamei, Alberto Rosi, and Franco Zambonelli. Supporting location-aware services for mobile users with the whereabouts diary. In *Proceedings of the 1st International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, pages 6:1–6:6. ICST, 2008.

[BDG+13] Tomás Bures, Rocco De Nicola, Ilias Gerostathopoulos, Nicklas Hoch, Michal Kit, Nora Koch, Giacoma Valentina Monreale, Ugo Montanari, Rosario Pugliese, Nikola B. Serbedzija, Martin Wirsing, and Franco Zambonelli. A life cycle for the development of autonomic systems: The e-mobility showcase. In *7th IEEE International Conference on Self-Adaptation and Self-Organizing Systems Workshops, SASOW, 2013, Philadelphia, PA, USA, September 9-13, 2013*, pages 71–76, 2013.

[BETVG08] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[BFZ14a] Nicola Bicocchi, Damiano Fontana, and Franco Zambonelli. Human aware superorganisms. In *The 2014 ACM Conference on Ubiquitous Computing, UbiComp '14 Adjunct, Seattle, WA, USA - September 13 - 17, 2014*, pages 1057–1062. ACM, 2014.

[BFZ14b] Nicola Bicocchi, Damiano Fontana, and Franco Zambonelli. Improving activity recognition via satellite imagery and commonsense knowledge. In *ISSASiM 2014 4th DEXA Workshop on Information Systems for Situation Awareness and Situation Management*, Munich, Germany, 2014.

[BMZ10] Nicola Bicocchi, Marco Mamei, and Franco Zambonelli. Detecting activities from body-worn accelerometers via instance-based algorithms. *Pervasive and Mobile Computing*, 6(4):482–495, 2010.

[BVZH14] Nicola Bicocchi, Emil Vasses, Franco Zambonelli, and Mike Hinchey. Reasoning on data streams: an approach to adaptation in pervasive systems. In *FMSAS 2014 2nd International Workshop on Formal Methods for Self-adaptive Systems*, Ho Chi Min Town, Vietnam, 2014.

[CCC+14] Giacomo Cabri, Nicola Capodieci, Luca Cesari, Rocco De Nicola, Rosario Pugliese, Francesco Tiezzi, and Franco Zambonelli. Self-expression and dynamic attribute-based ensembles in SCEL. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, October 8-11, 2014, Proceedings, Part I*, volume 8802 of *Lecture Notes in Computer Science*, pages 147–163. Springer, 2014.

[CCZ14] Nicola Capodieci, Giacomo Cabri, and Franco Zambonelli. Modeling self-expression by holons. In *International Conference on High Performance Computing & Simulation, HPCS 2014, Bologna, Italy, 21-25 July, 2014*, pages 424–431, 2014.

[CPZ11] G. Cabri, M. Puviani, and F. Zambonelli. Towards a taxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In *International Conference on Collaboration Technologies and Systems*, pages 508–515. IEEE, 2011.

[Dic06] Dave Dice. *Biased Locking in HotSpot*, 2006. `https://blogs.oracle.com/dave/entry/biased_locking_in_hotspot`.

[DMS10] D. Dice, M.S. Moir, and W.N. Scherer. Quickly reacquirable locks, October 12 2010. US Patent 7,814,488.

[DMSGK06] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. Self-organisation and emergence in mas: An overview. *Informatica*, 30:45–54, 2006.

[DMSGK11] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos, editors. *Self-Organising Software*. Natural Computing. Springer, 2011.

[DMSKRZ04] Giovanna Di Marzo Serugendo, Anthony Karageorgos, Omer F Rana, and Franco Zambonelli. Engineering self-organising systems: nature-inspired approaches to software engineering. *LNCS*, 2004.

[DNLPT14] Rocco De Nicola, Michele Loreti, Rosario Pugliese, and Francesco Tiezzi. A formal approach to autonomic systems programming: the scel language. *ACM Transactions on Autonomous and Adaptive Systems*, pages 1–29, 2014.

[DWH05] Tom De Wolf and Tom Holvoet. Emergence versus self-organisation: different concepts but promising when combined. In Sven A. Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal, editors, *Engineering Self-Organising Systems*, volume 3464 of *LNCS*, pages 1–15. Springer, 2005.

[FM11] Laura Ferrari and Marco Mamei. Discovering daily routines from google latitude with topic models. In *Proceedings of 11th IEEE International Conference on Pervasive Computingand Communications Workshops*, pages 397–402. IEEE Computer Society, 2011.

[GGC11] Jean-Pierre Georg, Marie-Pierre Gleizes, and Valrie Camps. Cooperation. In Di Marzo Serugendo et al. [DMSGK11], pages 193–226.

[HG03] Francis Heylighen and Carlos Gershenson. The meaning of self-organization in computing. *IEEE Intelligent Systems, Section Trends & Controversies*, 18(4):72–75, 2003.

[HHK+13] Vojtech Horky, Frantisek Haas, Jaroslav Kotrc, Martin Lacina, and Petr Tuma. Performance regression unit testing: Acasestudy. In MariaSimonetta Balsamo, WilliamJ. Knottenbelt, and Andrea Marin, editors, *Computer Performance Engineering*, volume 8168 of *Lecture Notes in Computer Science*, pages 149–163. Springer Berlin Heidelberg, 2013.

[HRJ08] Jon Hall, Lucia Rapanotti, and Michael Jackson. Problem-oriented software engineering: Solving the package router control problem. *Transactions on Software Engineering*, 34(2):226–241, 2008.

[LS04] H. Liu and P. Singh. Conceptnet, a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226, 2004.

[Mit09] Melanie Mitchell. *Complexity: A guided tour*. Oxford University Press, 2009.

[MS08] Pawel Majewski and Julian Szymański. Text categorization with semantic commonsense knowledge: First results. In Masumi Ishikawa, Kenji Doya, Hiroyuki Miyamoto, and Takeshi Yamakawa, editors, *Neural Information Processing*, pages 769–778. Springer-Verlag, 2008.

[NZ14a] Victor Noel and Franco Zambonelli. Engineering emergence in multi-agent systems: Following the problem organisation. In *International Conference on High Performance Computing & Simulation, HPCS 2014, Bologna, Italy, 21-25 July, 2014*, pages 444–451, 2014.

[NZ14b] Victor Nol and Franco Zambonelli. Following the problem organisation: A design strategy for engineering emergence (short). In *Proceedings of the 8th International Symposium on Intelligent Distributed Computing (IDC'2014), Madrid (Spain)*. Springer, 2014.

[Ora14] Oracle. *Garbage Collector Ergonomics*, 2014. `http://docs.oracle.com/javase/7/docs/technotes/guides/vm/gc-ergonomics.html`.

[PEC09] Ognen Paunovski, George Eleftherakis, and Tony Cowling. Disciplined exploration of emergence using multi-agent simulation framework. *Computing and Informatics*, 28(3):369–391, 2009.

[PHBG09] Gauthier Picard, Jomi Fred Hübner, Olivier Boissier, and Marie-Pierre Gleizes. Reorganisation and Self-organisation in Multi-Agent Systems. In *International Workshop on Organizational Modeling*, pages 66–80, 2009.

[PPC+13] Mariachiara Puviani, Carlo Pinciroli, Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Is self-expression useful? evaluation by a case study. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2013 IEEE 22nd International Workshop on*, pages 62–67. IEEE, 2013.

[RGH+07] Sebastian Rodriguez, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abderrafiâa Koukam. An analysis and design concept for self-organization in holonic multiagent systems. In *Engineering Self-Organising Systems*, pages 15–27. Springer, 2007.

[She11] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, 2011.

[ZBC+11] Franco Zambonelli, Nicola Bicocchi, Giacomo Cabri, Letizia Leonardi, and Mariachiara Puviani. On self-adaptation, self-expression, and self-awareness in autonomic service component ensembles. In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011 Fifth IEEE Conference on*, pages 108–113. IEEE, 2011.