# Autonomous Systems: from Requirements to Modeling and Implementation

Nikola Šerbedžija
Fraunhofer FOKUS
Berlin, Germany
nikola.serbedzija@fokus.fraunhofer.de

***Abstract*—Developing autonomous systems requires adaptable and context aware techniques. The approach described here decomposes a complex system into service components – functionally simple building blocks enriched with local knowledge attributes. The internal components' knowledge is used to dynamically construct ensembles of service components. Thus, ensembles capture collective behavior by grouping service components in many-to-many manner, according to their communication and operational/functional requirements. Linguistic constructs and software tools have been developed to support modeling, validation, development and deployment of autonomous systems. A strong pragmatic orientation of the approach is illustrated by two different scenarios.**

***Keywords-autonomous systems; component-based system; context-aware systems***

## I. INTRODUCTION

Developing massively distributed systems has always been a grand challenge in software engineering [1,2,3]. Incremental technology advances have continuously been followed by more and more requirements as distributed applications grew mature. Nowadays, one expects a massive number of nodes with highly autonomic behaviour still having harmonized global utilization of the overall system. Our everyday life is dependent on new technology which poses extra requirements to already complex systems: we need reliable systems whose properties can be guaranteed; we expect systems to adapt to changing demands over a long operational time and to optimize their energy consumption [4,5].

One engineering response to these challenges is to structure software intensive systems in ensembles featuring autonomous and self-aware behaviour [6,7]. The major objective of the approach is to provide formalisms, linguistic constructs and programming tools featuring autonomous and adaptive behavior based on awareness. Furthermore, making technical systems aware of the energy consumption contributes significantly to the ecological requirements, namely to save energy and increase overall system utilization. The focus here is to integrate the functional, operational and energy awareness into the systems providing autonomous functioning with reduced energy consumption. The rationale, expressing power and practical value of the approach are illustrated on e-mobility and cloud computing application domains. The two complex domains appear to be fairly different. However, taking a closer look at the requirements of the two scenarios it becomes noticeable that the problem domains share numerous generic system properties, especially seen from the optimized control perspective.

The paper presents work in progress focusing on energy optimization in complex distributed control systems. It further elaborates methods and techniques to model and construct complex distributed systems with service components and ensembles. The rationale of the approach is presented through close requirements analysis, system modeling and development. The deployment is illustrated by the science cloud application scenario. Finally, the approach is summarized giving further directions for the work to come.

## II. REQUIREMENTS ANALYSIS

To explore the system requirements, two complex application domains are closely examined: e-mobility control and cloud computing.

E-mobility is a vision of future transportation by means of electric vehicles network allowing people to fulfill their individual mobility needs in an environmental friendly manner (decreasing polution, saving energy, sharing vehicels, etc).

Cloud computing is an approach that delivers computing resources to users in a service-based manner, over the internet, thus re-inforsing sharing and reducing energy consumption).

At a first glance electric vehicular transportation and distributed computing on demand have nothing really in common!

### A. Common Characteristics

In a closer examination the two systems, though very different, have a number of common characteristics.

#### 1) Massive Distribution and Individual Interest

E-mobility deals with managing a huge number of e-vehicles that transport people from one place to another taking into account numerous restrictions that the electrical transportation means imposes.

Each cloud computing user has also his/her individual application demands and interest to efficiently execute it on the cloud. The goal of cloud computing is to satisfy all these competing demands.

Both applications are characterized with huge number of single entities with individual goals.

### 2) *Sharing and Collectiveness*

In order to cover longer distances, an e-vehicle driver must interrupt the journey to either exchange or re-charge the battery. Energy consumption has been the major obstacle in a wider use of electric vehicles. Alternative strategy is to share e-vehicles in a way that optimizes the overall mobility of people and the spending of energy. In other words: when my battery is empty – you will take me further if we go in the same direction and vice versa [8].

The processing statistics show that most of the time computers are idle – waiting for input to do some calculations. Computers belong amongst the fastest yet most wasteful devices man has ever made. And they dissipate energy too. Cloud computing overcomes that problem by sharing computer resources making them better utilized. In another words, if my computer is free – it can process your data and vice versa; or even better, let us have light devices and leave a heavy work for the cloud [9].

At a closer look “sharing and collectiveness” are common characteristics of both application domains!

### 3) *Awareness and Knowledge*

E-mobility can support coordination only if e-vehicles know their own restrictions (battery state), destinations of users, re-charging possibilities, parking availabilities, the state of other e-vehicles nearby. With such knowledge collective behavior may take place, respecting individual goals, energy consumption and environmental requirements.

Cloud computing deals with dynamic (re-)scheduling of available (not fully used) computing resources. Maximal utilization can only be achieved if the cloud is “aware” of the users’ processing needs and the states of the deployed cloud resources. Only with such knowledge a cloud can make a good utilization of computers while serving individual users’ needs.

At a closer look “awareness” of own potentials, restrictions and goals as well as those of the others is a common characteristic. Both domains require self-aware, self-expressive and self-adaptive behavior based on a knowledge about those “self*” properties.

### 4) *Dynamic and Distributed Energy Optimization*

E-mobility is a distributed network that manages numerous independent and separate entities such as e-vehicles, parking slots, re-charge stations, drivers. Through collective and awareness-rich control strategy the system may dynamically re-organize and optimize the use of energy while satisfying users’ transportation needs.

Cloud computing actually behaves as a classical distributed operating system with a goal to maximize operation and throughput and minimize energy consumption, performing tasks of multiple users.

At a closer look “dynamic and distributed optimization” is inherent characteristic of the control environment for both application domains.

TABLE I. COMMON CHARACTERISTICS

| ***Common feature*** | ***Cloud computing*** | ***E-Mobility*** |
|---|---|---|
| Single entity | Computing resource | Vehicle, driver, park place, charging station |
| Individual goal | Efficient execution | Individual route plan |
| Ensemble | application , cpu pool, | Free vehicles, free park places, etc |
| Global goal | Resource availability, optimal throughput | Travel, journey, low energy |
| Self-awareness | avail-able resources; computational requirements, etc | Awareness of own state and restrictions |
| Autonomous and collective behavior | Decentralized decision making, global optimization | Reaching all destinations in time, minimizing costs |
| Optimization | Availability, computational task execution | Destination achieve-ment in time, ve-hicle/infrastructure usage |
| Adaptation | According to avail-able resources | According to traffic, individual goals, in-frastructure, resource availability |
| Robustness | Failing resources | Range limitation, charging battery in-frastructure resources |

## B. *Common Approach*

This set of common features serve as a basis for modeling of such systems leading to a generic framework for developing and deploying complex autonomic systems. The table 1 summarized the common requirements that lead to four major behavioral principles: adaptation, self-awareness, knowledge and emergence.

## III. MODELING

Control systems for the two application domains have many common characteristics: they are highly collective, constructed of numerous independent entities that share common goals. Their elements are both autonomous and cooperative featuring a high level of self-awareness and self-expressiveness. A complex control system built out of such entities must be robust and adaptive offering maximal utilization with minimal energy and resource use.

Formal specification, programming and controlling of a complex massively parallel distributed system that features awareness, autonomous and collective behavior, adaptive optimization and robust functioning are grand challenges of computer science. These challenges, present in most of complex control systems, have served as motivation and inspiration for this approach [7].

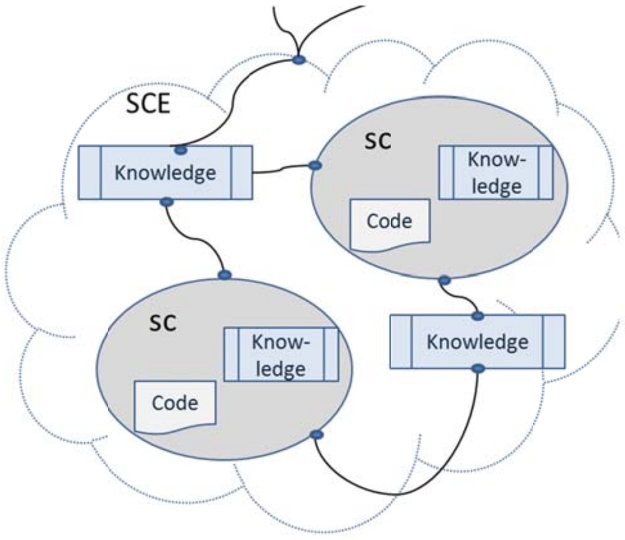

Figure 1. Service components and their ensebles

A complex system is decomposed in service components - major individual entities, and service component ensembles - composition structures that reflect communication and joint needs of service components:

- SC – service component are single system entities that have their requirements and functionality, usually representing their individual goals,
- SCE – service component ensembles are collections of service components usually representing collective system goals (as means to dynamically structure independent and distributed system entities).

The system structuring is depicted on Fig. 1.

Both components and ensembles have knowledge elements used to express their state and requirements. Based on this declarative knowledge, awareness, emergence and adaptive behavior can be achieved [7].

Fig. 2 illustrates an abstract view of modeling massively distributed systems with service components and ensembles. At the first level the real system entities are presented with different symbols representing different types of components. At the upper levels, different groupings are illustrated where components can be linked in ensembles, according to their requirements. There may be different ways of grouping, represented by different ensemble levels. One component can be a member of different ensembles at the same time. Ensembles are not fixed, during the system life time and according to the on-going states, re-grouping happens as a system response to dynamic changes.

## A. Modeling e-Mobility with Ensembles

Applying the general modeling strategy as depicted on Fig. 2 to e-mobility scenario, the different symbols at the first level could be interpreted as (1) users, (2) e-vehicles, (3) charging stations and (4) park places service components, where each component has knowledge on its

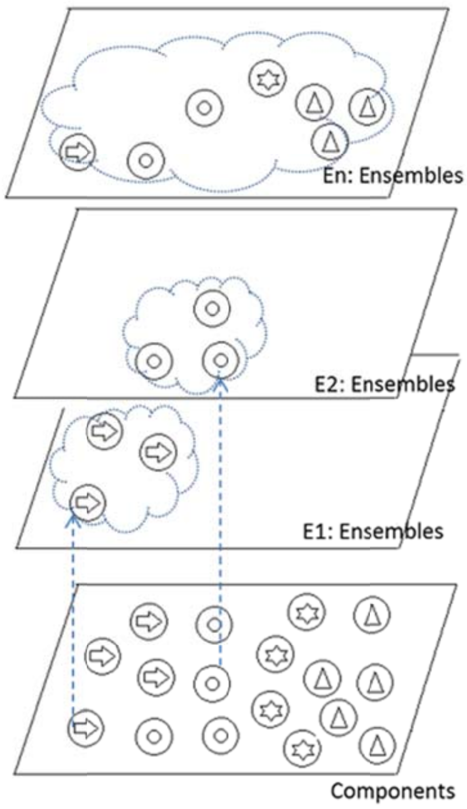

Figure 2. Modeling with components and ensebles

own state and needs. A user component knows the route plan having a goal to reach different places in a given time. A vehicle component has knowledge about its occupancy and battery state. Park places and charging stations maintain their availability/reservation plan. These major service components of the e-mobility scenario build the individual types with a huge number of instances. The E1and E2 ensemble levels show grouping according to the service component types, allowing users with nearby destinations to form an ensemble (with a common goal to reach the same destination and a possibility to share the vehicle) or vehicles with fully charged batteries at the same location to form an ensemble of available vehicles. The “En” ensemble level shows the e-mobility application with one user planning to use two vehicles, one parking place and a number of possible charging stations.

## B. Modeling Cloud Computing with Ensembles

In a similar manner, the same model shown on the Fig. 2 may represent an abstract cloud computing scenario. The major system elements represented by different symbols at the first level (E1) could be interpreted as (1) user applications, (2) remote computer CPUs, (3) local memory and (4) local application service components. Thereby, each component has knowledge about its own state and requirements. A user application component knows the requests for execution (in terms of CPU, minimal space, etc.). A remote computer component has knowledge about its processing capabilities and a current utilization. Disk components have knowledge of their capacity. Appi components have descriptions of the available appis at the local computer.

The E1and E2 ensemble levels show grouping according to the service component types, allowing e.g. grouping of appis of the same type with similar requests to form an ensemble or different CPUs to form an ensemble of available CPUs. The “En” ensemble level shows the cloud application with one user appi running at one remote CPU with a possibility to migrate to another CPU (with similar configuration), using one memory resource with a possibility to access a number of local applications.

Table 2 summarizes major service components within both application scenario mapping.

TABLE II. MAJOR SERVICE COMPONENTS

| *Symbols* | *E-Mobility* | *Cloud computing* |
|---|---|---|
| ⇨ | Users | User applications |
| ◎ | Electric vehicles | Remote computer CPUs |
| ✡ | Charging stations | Local memory |
| △ | Park places | Local application services |

### C. SCEL Language Programming Abstractions

The challenge for developers of complex distributed systems is to find proper linguistic abstractions to cope with individual vs. collective requirements of system elements and their need to respond to dynamic changes in an autonomous manner. A set of semantic constructs has been proposed [10,11] that represent behaviors, knowledge and composition supporting programming of awareness-rich system.

The basic ingredient of SCEL - Software Component Ensemble Language is the notion of autonomic component I[K;∏; P] that consists of:

- An interface I in a form of attributes – visible to other components.
- Knowledge repository K managing information about component interface, requirements, major state attributes etc. Managing such knowledge allows for self-aware behavior and dynamic interlinking with other system components.
- A set of policies ∏ that manage the internal and external interaction.
- A set of process P defines component functionality specific to both the application and internal management of knowledge, polices and communication.

The structure and organization of the SCEL notation is illustrated in Fig. 3,

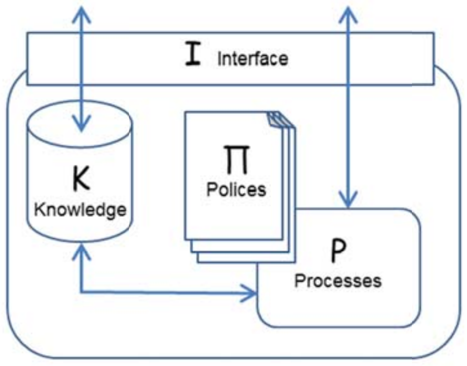

Figure 3. SCEL elememnts

Systems:
$S ::= C \mid S_1 \parallel S_2 \mid (\nu n)S$

Components:
$C ::= \mathcal{I}[\mathcal{K}, \Pi, \mathcal{P}]$

Processes:
$P ::= \mathbf{nil} \mid a.P \mid P_1 + P_2 \mid P_1[P_2] \mid X \mid A(p)$

Actions:
$a ::= \mathbf{get}(T)@c \mid \mathbf{qry}(T)@c \mid \mathbf{put}(t)@c \mid \mathbf{new}(\mathcal{I}, \mathcal{K}, \Pi, \mathcal{P})$

Targets:
$c ::= n \mid x \mid \mathbf{self} \mid P \mid \mathcal{I}.p$

The code above shows a fraction of SCEL syntax (with notation for S - systems, C - components, P - processes, a - actions and c - targets); a fully detailed presentation of SCEL syntax and semantics can be found in [10, 11].

The SCEL aggregates both semantics and syntax power to express autonomic behavior. At one side, being abstract and rigorous SCEL allows for formal reasoning about system behavior, at another, it needs further programming tools to support system development and deployment. Formal reasoning, modeling and validation are covered in referenced articles about SCEL. Here, the focus is more on pragmatic orientation on a given application scenario.

## IV. DEVELOPING AND DEPLOYING AUTNOMOUS SYSTEMS

A way from high level modeling to development and deployment of software intensive systems is a complex endeavor. Reasoning and validation often require high-level abstractions, while implementation calls for detailed programming and low-level deployments. To bridge this gap a number of intermediate tools are being developed that assist in the engineering process [7,12].

### A. Java Framework for SCEL Programming and Model Checking

To execute SCEL programs, the jRESP framework has been developed. This is a Java runtime environment providing means to develop autonomic and adaptive systems programmed in SCEL [13]. By relying on the jRESP API, a programmer can embed the SCEL paradigm in Java applications.

A prototype statistical model-checking running on top of jRESP simulation environment has been implemented. Following this approach, a randomized algorithm is used to verify whether the implementation of a system satisfies a specific property with a certain degree of confidence. The statistical model-checker is parameterized with respect to a given tolerance *t* and error probability *p*. The used algorithm guarantees that the difference between the computed values and the exact ones is greater than *t* with a probability lower than *p*.

The model-checker included in jRESP can be used to verify reachability properties. These properties allow one to evaluate the probability to reach, within a given deadline, a configuration where a given predicate on collected data is satisfied [13].

## B. Developing Science Cloud

Cloud computing is a modern paradigm for programming and utilizing distributed infrastructure resources in a dynamic way. Cloud-based systems are safety- and security-critical systems; they need to satisfy time-critical performance-based quality of service properties and to dynamically adapt to changes in the potentially hostile and uncertain environment they operate in. These aspects make distributed cloud-based systems complex and hard to design, build, test, and verify. The cloud scenario taken here is the cloud as a platform with voluntary peer-to peer configuration, meant to execute scientific applications [9]. It closely followed the modeling approach described in previous sections.

### 1) Service Components

Each instance of the Science Cloud Platform (SCP), running on a physical or virtual machine is considered to be a service component in the previous described sense. Fig. 4 shows the functionality required by a Science Cloud Platform instance. Two major characteristics of SCPs are further explored: knowledge and connectivity.

### 2) Knowledge

Each SCPi has knowledge consisting of (1) its own properties (set by developers), (2) its infrastructure (CPU load, available memory), and (3) other SCPis (acquired through the network). Since there is no global coordinator, each SCPi must build its own view and act upon the available knowledge. The SCPi may acquire knowledge about its infrastructure using an infrastructure sensing plug-in which provides information about static values, such as processor speed, available memory, available disk space, number of cores etc. and dynamic values, such as currently used memory, disk space, or CPU load.

SCPi properties are important when specifying conditions (Service Level Agreements, SLAs) for the applications. For example, when looking for a new SCPi to execute an application, low latency between the SCPs might be interesting. Other requirements may be harder: For example, an application may simply not fit on an SCPi because of the lack of space whereas another may require a certain amount of memory.

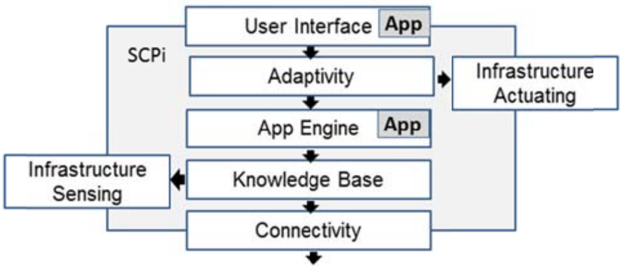

Figure 4. Science Cloud Framework

### 3) Connectivity

Each SCPi has a connectivity component which enables it to talk to other SCPs over the network. The protocol followed by these communications must enable SCPs to find one another and establish links, for example by manually entering a network address or by a discovery mechanism. Furthermore, SCPs must be able to query others for knowledge and at the same time distribute their own knowledge. Finally, the protocol must support exchange of data and applications.

Fig. 4 illustrates an instance of a science cloud platform as a part of a rich virtual framework for executing scientific applications. Through awareness of its own properties and those of others it offers maximal utilization of available computing resources within the cloud.

As already indicated an SCPi is adaptive and can react to conditions such as overload, shutdown of other SCPs, etc. Furthermore, it must watch over the apps executed and guarantee their SLAs (Service Level Agreement) [7,9]. This functionality is performed in an adaptivity logic component. The adaptivity logic is exchangeable, application-independent, and has a direct relation to the SLAs of applications. The adaptivity logic itself can be written in a standard programming language or custom domain-specific languages or rules.

Finally, each SCPi provides the application execution service to upper levels. The applications run on the platform must implement some API for the platform to be able to work with them (i.e. starting, stopping, working with data, etc.). One example of an app is the data storage service which allows users to store data in the cloud.

### 4) Ensembles

A Science Cloud Platform Ensemble (SCPe) consists of individual SCPs based on a set of properties of the SCPs and/or the SLAs of applications. In another words, an ensemble consists of SCPs which work together to run one application in a fail-safe manner and under consideration of the SLA of that application, which may require a certain number of SCPs, certain latency between the parts, or have restrictions on processing power or on memory.

At runtime, an ensemble may gain new SCPs or lose them depending on the behavior of the SCPIs themself and also on the load generated by the application itself or other applications running on the SCPIs.

### C. *Application Deployment*

Currently, a prototype of a science cloud platform is being developed and tested in a physical network connecting two universities [7]. The experimental platform does feature ad hoc and voluntary behavior supporting dynamic re-configuration of physical layers and application migration on an upper level. High-level SCEL modeling and model checking provide formal means for properties proofs while a prototype implementation offers pragmatic means to test deployment and effectiveness of autonomous and self-aware behavior.

## V. CONCLUSION

The paper presents a unified approach to model, validate and deploy complex distributed systems with massive number of nodes that respect both individual and global goals. Non-centralized character of the approach allows for autonomic and self-aware behavior, which is achieved by introduction of knowledge elements and enrichment of compositional and communication primitives with awareness of both system requirements and individual state of the computing entities.

The essence of the approach is to de-compose a complex system into a number of generic components and to further compose the system into ensembles of service components.

The inherent complexity of ensembles is a huge challenge for developers. Thus, the whole system is decomposed into well-understood building blocks, reducing the innumerable interactions between low-level components to a manageable number of interactions between these building blocks. The result is a so-called hierarchical ensemble, built from service components, simpler ensembles and knowledge units connected via a highly dynamic infrastructure. Ensembles exhibit four main characteristics: adaptation, self-awareness, knowledge and emergence, yielding a sound technology for engineering autonomous systems [5,7]. A number of linguistic constructs and validation and programming tools are under development and are being tested in different application scenarios.

This paper presents an integrated view (from high level modeling to application deployment) of a complex approach which has been described by a number of referenced papers, each focusing on different aspects of the work: SCEL modeling [10,11] and system validation [13], adaptation aspects[8], knowledge management and deployments [8,9] and engineering aspects [5,7]. Further contribution of this paper is in optimized control based on awareness and autonomous behavior.

Optimized distributed control with improved throughput and utilization of the cloud and e-mobility frameworks contribute significantly to the overall strategy to reduce energy consumption. Sharing principle instead of exclusive use of the computing and transportation means represent a significant challenge (requiring significant changes in our perception of vehicles and computers) in the application domains under consideration. This principle will undoubtedly play an important role in extending the application domains.

## ACKNOWLEDGMENT

Most of the work presented here has been done under the ASCENS project (project number FP7- 257414) [7], funded by the European Commission within the 7th Framework Programme, pervasive adaptation initiative. Special thanks go to the developers of SCEL language (Rocco De Nicola from IMT Lucca and his group) and the developers of cloud computing application (Philip Mayer and the whole ASCENS team from LMU Munich).

## REFERENCES

[1] Project InterLink: http://interlink.ics.forth.gr [retrieved: Feb.2013].

[2] I. Sommerville et al., “Large-scale complex it systems”. Commun. ACM, vol.55, no.7, 2012, pp.71-77.

[3] M. Hoelzl, A. Rauschmayer, and M. Wirsing, “Engineering of software-intensive systems”, in Wirsing, M., Banatre, J.P., Hoelzl, M., Rauschmayer, A., eds.: Software-Intensive Systems and New Computing Paradigms. Vol.5380 of LNCS, 2008, pp.1-44.

[4] L. Xu, G. Tan, X. Zhang, and J. Zhou, “Energy aware cloud application management in private cloud data center”, 2011 International Conference on Cloud and Service Computing, 2011, pp.274-279.

[5] C. Seo, “Energy-Awareness in Distributed Java-Based Software Systems”, 21st IEEE International Conference on Automated Software Engineering (ASE'06), 2006, pp.343-348.

[6] M. Hoelzl and M. Wirsing, “Towards a system model for ensembles”, in G. Agha, O. Danvy, and J. Meseguer (eds.), Formal Modeling: Actors, Open Systems, Biological Systems, Lecture Notes in Computer Science Vol.7000, 2011, pp. 241–261.

[7] Project ASCENS (Autonomic Service-Component Ensembles), http://www.ascens-ist.eu ASCENS, 2012, [retrieved: Feb.2013].

[8] D. Abeywickrama, F. Zambonelli, and N. Hoch, “Towards Simulating Architectural Patterns for Self-Aware and Self-Adaptive Systems”, In 2nd SASO Workshop on Awareness in Autonomic Systems, Lyon (F), 2012, pp.87-94.

[9] P. Zormeier, A. Klarl, C. Kroiss, and P. Mayer, “Science Cloud: Modelling and Implementing the Peer-to-Peer DHT protocol ’Chord’”, Technical Report, Ludwig-Maximilians-Universitt Mnchen, Germany, 2012.

[10] R. De Nicola, G-L. Ferrari, M. Loreti, and R. Pugliese, “A Language-based Approach to Autonomic Computing”, In Formal Methods for Components and Objects, vol.7542 Lecture Notes in Computer Science, 2012, pp.26-48.

[11] R. De Nicola, M. Loreti, R.Pugliese, and F. Tiezzi, “SCEL: a language for autonomic computing”, Technical Report, [retrieved: Feb.2013].

[12] M.P. Ashley-Rollman, S.C. Goldstein, P. Lee, T.C. Mowry, and P. Pillai, “Meld: A declarative approach to programming ensembles”, In: IROS, IEEE, 2007, pp.2794-2800.

[13] M. Loreti. jRESP: a run-time environment for scel programs, Technical Report, http://rap.dsi.unifi.it/scel/, [retrieved: Feb.2013].