Constructing Autonomous Multi-Robot System

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

Abstract—Developing control systems for swarm robotics require advanced techniques that can ensure adaptive, autonomous, self-aware and intelligent behavior. An engineering response to such demands is an ensemble based approach that structures a complex control system into dynamic ensembles of relatively simple system elements, called service components. The dynamism and autonomous behavior of the system elements are modeled by the knowledge- and predicate-based communication principle that allows for late (at run-time) evaluation of communication and connection rules among the system elements. The approach is illustrated on a concrete multi-robot scenario.

Keywords--swarm robotic; autonomous systems; development life cycle, ensemble-based system.

I. MOTIVATION

Constructing a multi-robot system requires multidisciplinary approach that calls for advanced techniques from the domains of software engineering, parallel and distributed system, agent systems and artificial intelligence. Each of the target disciplines poses grand challenges in its own field [1][2]. To respond to changing demands over a long operational time, adaptive and autonomous behavior at both individual and collective level [3] as well as energy awareness [4][5] need to be ensured.

The solution offered here responds to all these challenges. The approach decomposes a complex system into high number of service components – functionally simple building blocks enriched with knowledge attributes [6]. The knowledge of a component controls autonomic behavior at a local level. To ensure meaningful grouping and autonomy at higher levels (collective autonomy), system components are grouped into ensembles according to predicates over the components' attributes (which represents the major novelty of the approach). These predicates are actually implicit rules for communication bindings and represent global knowledge of the ensemble.

In order to guarantee correct and timely behavior in such demanding circumstances, this approach relies on formal methods. The system design and development phases are strictly defined leading to step-wise process of modelling, development, verification and validation.

The emphasis of this paper is on major engineering phases of the ensemble development lifecycle. A strongly pragmatic approach is illustrated by the concrete multi-robot scenario.

The paper is structured into six sections describing motivation (section one), engineering approach (section two), problem description (section three), system modelling using the SCEL language and JRESP framework (section four) and the deployment (section five). The conclusion (section 6) summarizes the achievements and indicates further directions for the work to come.

II. ENGINEERING APPROACH

Autonomous systems introduce a number of requests which are not present in other less dynamic systems. Constant changes both in the controlled environment and in the system *per se* require an appropriate methodology. The development process needs to be continuous, allowing for re-consideration and refinement both during the system development and during the system execution time. The approach described here proposes a persistent process for ensemble construction that consists of two major development circles, each having three phases:

- Design circle consists of:
  1. Requirement analysis,
  2. Modelling and programming, and
  3. Validation and verification phases.
- Runtime circle contains of
  1. Monitoring,
  2. Awareness, and
  3. Self-adaptation phases.

Two transitions, namely deployment and feedback ensure the correlation among the two circles.

- Deployment is a step-wise transition that is the result of modeling and programming phases. It begins with the first release and later continues whenever system modification occurs (re-deployment).
- Feedback is a transition that represents re-engineering, i.e., a system modification caused by problems discovered within the monitoring, awareness or self-adaptation run-time phases.

To ensure rigorous development of complex distributed autonomous systems, a number of tools and methods have been developed to support each of the phases and transitions within the development life cycle [7]. This paper focuses on tools and methods for modelling and the deployment, namely the SCEL (Service Component Ensemble Language) [8], and the jRESP (Java Runtime Environment for SCEL Programs) [9] and ARGoS[10] frameworks.

III. PROBLEM SPECIFICATION

Swarm robotics deals with creation of multi-robot systems that through interaction among participating robots and their environment can accomplish a common goal, which would be impossible to achieve by a single robot. To illustrate the

 ISBN: 978-1-61208-352-0

application from the swarm robotics domain a search and rescue scenario is presented.

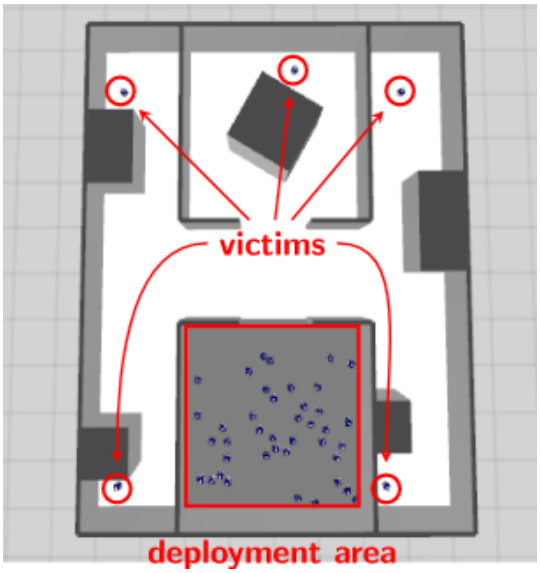


Figure 1. Scenario

A. *Swarm Robotics Scenario*

The basic idea behind the scenario is to organize and control a rescue operation in an emergency situation. Figure 1 illustrates the scenario where "an explo-sion happens in a nuclear plant causing the radiation, spill and collapse of a part of the building where a number of victims is trapped. To prevent further harm to human lives, a team of robots is deployed in the endangered area. The robots must explore the area, search for victims, and coordinate to save the victims as fast as possible. Besides removing victims, robots have to neutralize the radiation source by building blocks around it".

In the above scenario, a swarm of robots is distributed in a so called deployment area. The robots must reach the zone according to the scenario goal (finding victims and radiation source, carrying blocks, etc). Robots are not informed about the position of the targets. To discover their location they perform random walk combined with coordinated exploration. As soon as a robot reaches a radiation zone or a victim, it 'publishes' its location within the local knowledge repository. In this way, robots with the same task can be informed about the location of the corresponding target. Informed robots can then move directly towards the target thus saving time and energy.

Robots possess limited battery lifetime. To behave in an energy-aware manner, the robots must monitor the battery charge over the course of the experiment. If the battery charge drops too low, self-healing actions are required, e.g., reaching a charging station or sending a distress signal.

There are two types of robots in a multi-robot system needed to solve the "search and rescue" problem, as specified in the given scenario (see Figure 1): a - foraging robots that explore the environment and find objects and b - robots with a gripper, which can carry objects.

a) Foraging robot b) Robot with a gripper

Figure 2, Swarm robots

B. *Generic System Properties*

To further explore the control system requirements, the given scenario is closely examined and the major system characteristics are extracted (formulated in a generic form in order to keep them applicable in other application scenarios):

1. Individual goals
2. Coordination and distribution
3. Sharing and collectiveness (global goals)
4. Awareness and knowledge
5. Energy awareness and optimization

Each robot from the swarm has an individual goal (ie. simple task it can do). To solve a collective assignment, robots dynamically gather in a swarm, which further requires coordinated and distributed behavior. Knowledge of own capabilities and conditions as well as of those from the environment, bring awareness at both local and global level. Throughout its operational time, each robot from a swarm needs to observe its battery state and to adapt its functioning appropriately.

In a summary, a typical swarm robotics control system is highly collective, constructed of numerous independent entities that share common goals. Its elements are both autonomous and cooperative featuring a high level of self-awareness, self-expressiveness.

C. *Specific Scenario Properties*

In order to accomplish the rescue mission from the given scenario the robots need to perform the following operations: (i) efficient operation as robot energy depletes, and (ii) reaching consensus on the order in which the victims must be saved.

 ISBN: 978-1-61208-352-0

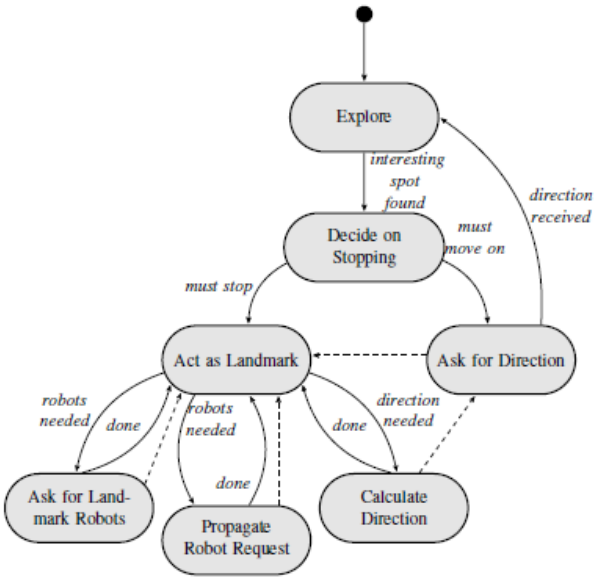


Figure 3. Robot behaviour graph

To solve the rescue scenario, a specific robot behavior called distributed exploration, is further specified. Robots are divided in two groups: workers and landmarks. Workers are robots that perform the actual rescuing task, transporting the victims to the deployment area. Landmark robots explore and mark important locations in the environment. Landmark robots are deployed first. They exit the deployment area one by one, moving straightforward until they encounter either a branching or an important location (e.g., a victim), or they are about to lose connectivity with the previous robots. Landmark robots form a network that is used by next approaching landmark robots.

Figure 3 shows a behavior graph of the "landmark" robot from a swarm. Based on the graph, the robot behavior is further specified, modeled, simulated and finally deployed on real robots.

IV. MODELING AND PROGRAMMING

Valid modeling and programming techniques ensure later correct behavior. The ensemble development lifecycle (EDLC) [7] uses a rigorous modeling/programming approach that allows for both formal reasoning on system properties and semi-automatic programming and validation.

A control system is decomposed into simpler hierarchical elements [9] called service components (SC) - representing simple functional entities with clearly defined individual goals, and service component ensembles (SCE) - representing a collection of service components with clearly defined collective goals.

Both components and ensembles have local knowledge used to express their goals. Knowledge is represented in terms of system properties and the goals are attributes over these properties.

A. Modeling Language SCEL

The basic entity of SCEL - Software Component Ensemble Language is the notion of autonomic component $I[K; \Pi; P]$ that consists of the following elements:

- An interface I given in a form of attributes – visible to other components.
- Knowledge repository K containing 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 Π that manage the internal and external interaction.
- A set of processes P defines component functionality specific to both the application and the internal management of knowledge, polices and communication.

For specification of processes, SCEL features a process algebra, which is extended by knowledge manipulation actions: **get** – taking a knowledge field out of the knowledge repository (blocks if not present), **qry** – getting a value of knowledge field while keeping the field in the knowledge repository (blocks if not present), **put** – inserting a knowledge field into the knowledge repository. The knowledge manipulation actions may use direct addressing (including a special target **self**) as well as addressing using a predicate, in which case, the action is performed on the knowledge of all components that matches the predicate (implicitly, creating an ensemble). A fully detailed presentation of SCEL syntax and semantics can be found in [7][8].

B. Modeling the Robot Scenario in SCEL

Qualitatively, the behavior of a single robot could be modeled with the following SCEL fragment, where each component $I[K; \Pi; (AM[ME])\ ]$ has the following description:

$$ME \triangleq \mathbf{qry}(\text{"controlStep"}, ?X)@\mathbf{self}.(\mathbf{get}(\text{"termination"})@\mathbf{self}.ME)[X]$$

$$AM \triangleq P_{batteryMonitor}[P_{dataSeeker}[P_{targetSeeker}]]$$

The processes composing the autonomic manager AM are as follows:

$$\begin{aligned} P_{batteryMonitor} \triangleq\ & \mathbf{qry}(\text{"batteryLevel"}, \text{"low"})@\mathbf{self}. \\ & \mathbf{get}(\text{"lowBattery"}, \text{false})@\mathbf{self}. \\ & \mathbf{put}(\text{"lowBattery"}, \text{true})@\mathbf{self}. \\ & \mathbf{qry}(\text{"batteryLevel"}, \text{"high"})@\mathbf{self}. \\ & \mathbf{get}(\text{"lowBattery"}, \text{true})@\mathbf{self}. \\ & \mathbf{put}(\text{"lowBattery"}, \text{false})@\mathbf{self}.P_{batteryMonitor} \end{aligned}$$

$$\begin{aligned} P_{dataSeeker} \triangleq\ & \mathbf{qry}(\text{"targetLocation"}, ?x, ?y)@(\mathcal{I}.task = \text{"task}_i\text{"}). \\ & \mathbf{put}(\text{"targetLocation"}, x, y)@\mathbf{self}. \\ & \mathbf{get}(\text{"informed"}, \text{false})@\mathbf{self}. \\ & \mathbf{put}(\text{"informed"}, \text{true})@\mathbf{self} \end{aligned}$$

Furthermore, a foraging robot (*TargetSeaker*) is described as:

$$
\begin{aligned}
P_{targetSeeker} \triangleq\ & \mathbf{qry}(\text{“}lowBattery\text{”}, ?low)@\mathsf{self}. \\
& \textbf{if } (low) \textbf{ then } \{ \\
& \quad \mathbf{get}(\text{“}controlStep\text{”}, ?X)@\mathsf{self}. \\
& \quad \mathbf{put}(\text{“}controlStep\text{”}, P_{lowBattery})@\mathsf{self}. \\
& \quad \mathbf{qry}(\text{“}lowBattery\text{”}, \mathsf{false})@\mathsf{self}. P_{targetSeeker} \\
& \} \textbf{ else } \{ \\
& \quad \mathbf{qry}(\text{“}target\text{”}, ?found)@\mathsf{self}. \\
& \quad \textbf{if } (found) \textbf{ then } \{ \\
& \quad\quad \mathbf{get}(\text{“}controlStep\text{”}, ?X)@\mathsf{self}. \\
& \quad\quad \mathbf{put}(\text{“}controlStep\text{”}, P_{found})@\mathsf{self}. P_{doTask} \\
& \quad \} \textbf{ else } \{ \\
& \quad\quad \mathbf{qry}(\text{“}informed\text{”}, ?informed)@\mathsf{self}. \\
& \quad\quad \textbf{if } (informed) \textbf{ then } \{ \\
& \quad\quad\quad \mathbf{get}(\text{“}controlStep\text{”}, ?X)@\mathsf{self}. \\
& \quad\quad\quad \mathbf{put}(\text{“}controlStep\text{”}, P_{informed})@\mathsf{self}. P_{targetSeeker} \\
& \quad\quad \} \textbf{ else } \{ \\
& \quad\quad\quad \mathbf{get}(\text{“}controlStep\text{”}, ?X)@\mathsf{self}. \\
& \quad\quad\quad \mathbf{put}(\text{“}controlStep\text{”}, P_{randomWalk})@\mathsf{self}. P_{targetSeeker} \\
& \quad\quad \} \\
& \quad \} \\
& \}
\end{aligned}
$$

The autonomic behavior of each robot is realized by means of an autonomic manager (AM) controlling the execution of a managed element (ME). The autonomic manager monitors in a self-aware fashion the state of charge of a robot's battery and verifies whether the target area has been reached or not. Self-adaptation can be naturally expressed in SCEL by exploiting its higher-order features, namely the capability to store/retrieve (the code of) processes in/from the knowledge repositories and to dynamically trigger execution of new processes. The autonomic manager can replace the control step code from the knowledge repository, thus implementing the adaptation logic and changing the managed element's behavior. For example, when a robot becomes informed, it self-adapts (i.e., self-configures) through its autonomic manager in order to move directly towards the target area.

C. *Simulation and Validation in jRESP*

The jRESP [9] framework is a runtime environment that provides Java programmers with ability to develop autonomic and adaptive systems based on the SCEL concepts. SCEL identifies the linguistic constructs for modeling the control of computation, the interaction among possibly heterogeneous components, and the architecture of systems and ensembles. jRESP provides an API that permits using the SCEL paradigm in Java programs.

The architecture of a generic jRESP node is shown in Figure 4. Each node is executed over a virtual machine or a physical device that provides the access to input/output devices and to network connections. Each node aggregates a knowledge repository, a set of running processes/threads, and a set of policies. Structural and behavioral information about a node can be collected into an interface via a set of attribute collectors. Nodes interact through ports supporting both point-to-point and group-oriented communications.

The robot scenario modeled in SCEL (as described in the previous section) is programmed in jRESP in the following way. The process ME (managed element) is

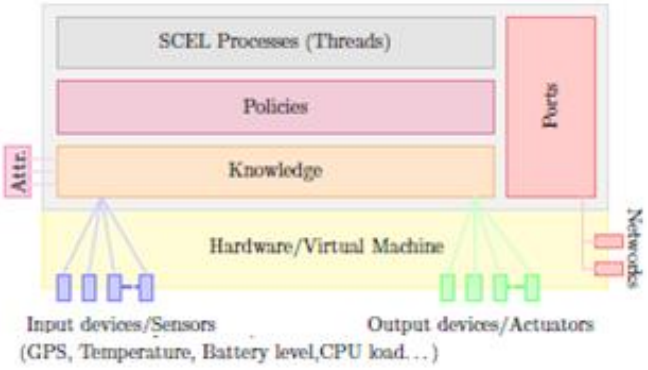


Figure 4. JRESP Architecture

rendered as an agent that continuously executes the control steps retrieved from the local knowledge repository:

```
public class ManagedElement extends Agent {
  public ManagedElement() {
    super("ManagedElement");
  }
  protected void doRun() throws Exception {
    while (true) {
      Tuple t = query(
        new Template(
              new ActualTemplateField("controlStep") ,
              new FormalTemplateField(Agent.class)) ,
         Self.SELF );
      Agent X = t.getElementAt(Agent.class, 1);
      X.call ();
    }
  }
}
```

The autonomic manager is modeled by the following three classes that provide a Java implementation for processes *P-batteryManager* and *P-dataSeeker* and *P-targetSeeker*, respectively:

```
public class BatteryMonitor extends Agent {
  public BatteryMonitor() {
    super("BatteryMonitor");
  }
  protected void doRun() throws IOException, InterruptedException{
     while (true) {
        query( new Template(
                  new ActualTemplateField("batteryLevel") ,
                  new ActualTemplateField("low")) ,
               Self.SELF );
        get( new Template(
                  new ActualTemplateField("lowBattery") ,
                  new ActualTemplateField(false)) ,
               Self.SELF );
        put( new Tuple( "lowBattery" , true ) , Self.SELF );
        query( new Template(
                  new ActualTemplateField("batteryLevel") ,
                  new ActualTemplateField("high")) ,
               Self.SELF );
        get( new Template(
                  new ActualTemplateField("lowBattery") ,
                  new ActualTemplateField(true)) ,
               Self.SELF );
        put( new Tuple( "lowBattery" , false ) , Self.SELF );
     }
   }

}
```

```
public class DataSeeker extends Agent {
  public DataSeeker() {
    super("DataSeeker");
  }
  protected void doRun() throws IOException, InterruptedException{
    Tuple t = query(new Template(
                        new ActualTemplateField("targetLocation") ,
                        new FormalTemplateField(Double.class),
                        new FormalTemplateField(Double.class)),
                    new Group(new HasValue( "task" , 1 ) ));
    double x = t.getElementAt(Double.class, 1);
    double y = t.getElementAt(Double.class, 2);
    put( new Tuple( "targetLocation" , x , y ) , Self.SELF );
    get( new Template(
                new ActualTemplateField("informed") ,
                new ActualTemplateField(false)) , Self.SELF );
    put( new Tuple( "informed" , true ) , Self.SELF );
  }
}

public class TargetSeeker extends Agent {
  public TargetSeeker() {
    super("DataSeeker");
  }
  protected void doRun() throws IOException, InterruptedException{
    while (true) {
```

A screen dump of a jRESP simulation of the robotic scenario is shown on Figure 5a, illustrating the movements of the foraging robots with a landmark searching algorithm.

Formal modelling of the multi-robot scenario also contributes to the validation phase of the software development lifecycle. As shown on figure 5b, the jRESP simulation can be used to calculate the probability of finding victim in the given scenario (for the given algorithm, the probability of success is directly proportional to the number of landmarks used in a search). The verification of the search algorithm is shown on the figure 5c, insuring that the algorithm will always converge. The simulation, validation and verification tools all refer to the problem described in the scenario shown on the Figure 1.

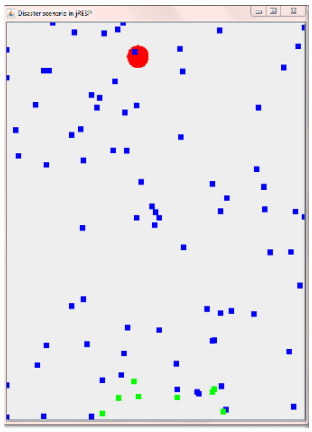
V. DEPLOYMENT

The deployment transition of the ensemble development life cycle involves the implementation of the robot behaviors on real robots. This step is the most critical in robotics because it is usually the most expensive, time-consuming, and risky. For this reason, deployment is usually performed in two distinct phases. The first phase consists of testing the robot behaviors in accurate physics-based simulations. These simulations must include as many details as possible, so as to minimize costly issues in the next phase. The next deployment phase consists of testing the behaviors on the real platform with robots.

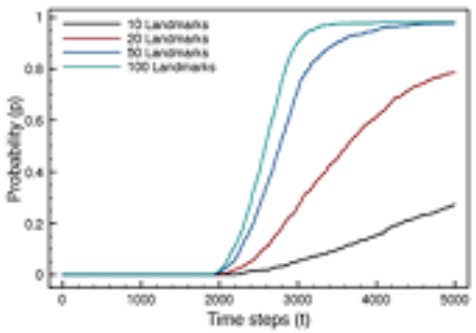
For the deployment purposes the ARGoS (discrete-time simulator for multi-robot systems) [11] platform is used as it provides both an efficient simulation framework and a straightforward deployment with real robots. The same control system is firstly tested on a simulated environment and then is transferred to the real platform, substituting simulated robots with the real ones..

ARGoS is a physics-based multi-robot simulator. It aims to simulate complex experiments involving large swarms of robots of different types in the shortest possible time. It is designed around two main and often contradictory requirements: efficiency - achieving high performance with large swarms, and flexibility - allowing the user to customize the simulator for specific experiments.

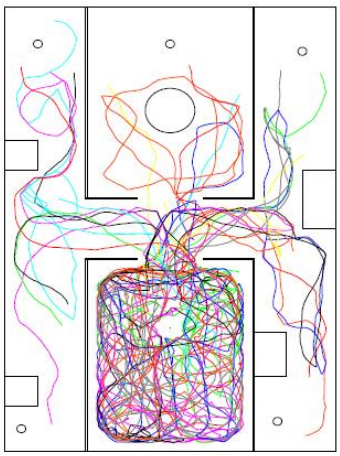
To bridge the efficiency and flexibility gap, ARGoS system deploys a number of novel design choices. First, in ARGoS, it is possible to partition the simulated space into multiple sub-spaces, managed by different physics engines running in parallel. Second, ARGoS' architecture is multi-threaded, thus designed to optimize the usage of modern multi-core CPUs. Finally, the architecture of ARGoS is highly modular. It is designed to allow the user to easily add



a) Simulation with foraging and the landmark robots



b) System validation



c) System verification

Figure 5. Screen dumps from the simulation, validation and verification tools

 ISBN: 978-1-61208-352-0

custom features (enhancing flexibility) and to allocate computational resources where needed (thus decreasing run-time and enhancing efficiency).

The final deployment phase in a real robot setting is still being developed. In preparation for the final deployment, simultaneously with ARGoS simulation, the two types of robots have been further refined (Figure 2).

VI. CONCLUSION

This paper presents an integrated approach to model, validate and deploy ensemble-based multi-robot systems. The 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 ensemble-based approach is to de-compose a complex system into a number of generic components, and then compose the system into ensembles of service components. The inherent complexity of such 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, providing a sound methodology for engineering autonomous systems. A number of analyses, modeling, programming and validation tools are under development and evaluation in different application settings [7].

The pragmatic significance of the approach has been illustrated by the multi-robot scenario showing the major design and development phases on the concrete practical example. The SCEL language [8] and jRESP [9] are used for modeling, programming and validating the scenario. Finally, ARGOS system [10] is used to fine-tune and deploy the control system in a real robot setting.

Further work is oriented towards monitoring and testing of the real system as well as towards analyses of the run-time behavior. These activities belong to the second cycle of the EDLC [7] and will be the subject of future work. Tools to monitor ensemble based systems should be developed that allow for run-time analyses and verification of awareness and self-adaptive behavior of both system elements and the system as a whole.

ACKNOWLEDGMENTS

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. Special thanks go to Roco de Nicola (CNR), for the work on SCEL [8], Michele Loreti (University of Florence) for the work on jRESP [9], and Carlo. Pinciroli (Brussels University) for the work on ARGoS[10].

REFERENCES

[1] I. Sommerville et al. Large-scale complex it systems. Commun. ACM. 2012, Vol.55, No.7, pp.71-77

[2] M. Hoelzl, A. Rauschmayer, and M. Wirsing. Engineering of software-intensive systems. In Software-Intensive Systems and New Computing Paradigms, LNCS, 2008, Vol.5380, pp.45-63.

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz. Swarm Intelligence: From Natural to Artificial Systems, New York, NY: Oxford University Press, Santa Fe Institute Studies in the Sciences of Complexity, 1999, ISBN 0-19-513159-2.

[4] B. Degener, B. Kempkes, and F. Meyer. Energy-Awareness in Self-organising Robotic Exploration Teams .Organic Computing, Springer, 2011, pp346-365.

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

[6] M. Hoelzl et al. Engineering Ensembles: A White Paper of the ASCENS Project. ASCENS Deliverable JD1.1. http://www.ascens-ist.eu/whitepapers [retrieved: May 2014].

[7] Project ASCENS (Autonomic Service-Component Ensembles). http://www.ascens-ist.eu ASCENS [retrieved: May 2014].

[8] R. De Nicola, M. Loreti, R. Pugliese, and F. Tiezzi. SCEL: a language for autonomic computing. Technical Report. Universita a degli Studi di Firenze. Available at: http://rap.dsi.unifi.it/scel/ [retrieved: May 2014].

[9] M. Loreti, jRESP: a run-time environment for scel programs. Technical Report. Universita a degli Studi di Firenze. Available at: http://rap.dsi.unifi.it/scel/, [retrieved: May 2014].

[10] C. Pinciroli et al. ARGoS: a Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems. Swarm Intelligence, Springer, Berlin, Germany, 2012, vol. 6, no. 4, pp 271-295.

 ISBN: 978-1-61208-352-0