# Mission-oriented Autonomic Configuration of Pervasive Systems

Guillaume Grondin, Matthieu Faure, Christelle Urtado and Sylvain Vauttier
*LGI2P / Ecole des Mines d'Alès, Nîmes, France*
*{Guillaume.Grondin, Matthieu.Faure, Christelle.Urtado, Sylvain.Vauttier}@mines-ales.fr*

***Abstract*—**In pervasive systems, software applications are dynamically composed from the services provided by the smart devices spread in the local environment. A system must react to changes that occur in the environment and reconfigure applications in order to maintain their operation and assume their missions at its best. This paper advocates the need for a mission description language, which enables to describe applications in a declarative way as abstract service compositions. The system uses mission definitions to calculate a configuration that best executes them with the currently available resources. This optimal configuration is intended to maximize the utility of the system, considering user preferences, available resources, and mission criticality. Contextual adaptations are captured in the mission language as modes and strategies, that respectively describe evolutions of the assigned mission set and alternate ways to execute missions. These mechanisms leverage service component approach, for the dynamic deployment of missions, and agent-orientation, for autonomic configuration management.

***Keywords*-**pervasive system; autonomic computing; context-awareness; service composition language.

## I. Introduction

As miniaturization of computer systems increases, calculation and communication-enabled (so-called smart) devices spread around us. These new computers altogether form computer networks that have given birth to a new paradigm called pervasive computing. Pervasive computing leverages the services that are dynamically discovered in the environment to meet user requirements. The dynamicity and openness of the environment have a strong impact on the software that must adapt to these changes. Thus, pervasive systems must provide solutions for several inherent issues [1]:

**- Context-awareness.** The boundaries and constituents of pervasive systems are not known beforehand. Pervasive software must be able to dynamically discover which resources are available in the environment. Moreover, it must know how to map user requirements with the available resources in order to implement valuable scenarios. Context-awareness is the capacity of pervasive software to sense and build a proper inner representation of varying environments.

**- User empowerment.** Pervasive systems are inherently user-oriented. Their purpose is to help users leverage the services provided by the smart devices that surround them in their environments. To be as useful and relevant as possible, pervasive systems must provide users with means to define and submit their own service compositions. Indeed, elaborated specific requirements cannot be met by predefined services. Pervasive systems must therefore support the dynamic definition of their missions, which entails dynamic adaptation touser demands.

**- Autonomous and dynamic adaptability.** As devices can freely join or quit the system anytime, resources and services are volatile. Pervasive systems must support dynamic change management to adapt themselves to open, variable environments. For dependability's sake, service continuity must be transparently maintained thus necessitating the system to autonomously and dynamically react in order to evaluate the situation and change either means used to reach its defined objectives or its objectives themselves..

The remaining sections of this paper describe the pervasive system framework we designed to tackle these issues. Section II sets the technical ground of our proposal, which is a combination of a multi-agent system and a service component architecture. It also briefly introduces a water hazard monitoring system as an application for our framework. Section III introduces the concepts and syntax of Arold, our proposed mission description language. Section IV briefly explains the principles of optimized deployment calculation. Section V discusses related work. Section VI draws a conclusion and perspectives about this work.

## II. Technical Ground and Case Study

### A. System, Agents, Components and Services

A pervasive system is dynamically composed of a set of smart devices co-located in an environment. Smart devices use their embedded communication capabilities and intelligence (software) to interact and cooperate. In our work (*see* Figure 1), the embedded software of each device is managed in a modular and reconfigurable way, which conforms to a service component approach [2].

Components are reusable, decoupled software modules that encapsulate distinct functionalities. They can be assembled together, so as to produce operational software, thanks to well defined connectable interfaces that explicitly document the interaction capabilities of components. Provided (server) interfaces document functionalities that are implemented by a component and proposed as

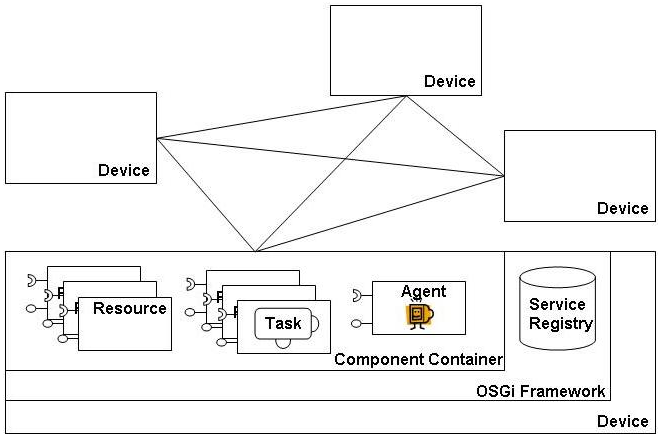


Figure 1. A component-based and autonomic pervasive system

invocable services. Required (client) interfaces document functionalities that must be called by a component as external services it depends on. We use the component model and container facility proposed by the OSGi framework [3] to dynamically administrate the installation, assembling, replacement or removal of components on each device. The OSGi framework also provides a service registry where any instantiated component can publish the services it proposes through its provided interfaces. This enables the dynamic introspection and retrieval of components that are available in a container. Components can thus be managed and handled so as to build and adapt software to the changing contexts in which devices must operate. Three kinds of components are actually deployed on a device: resource components, that provide low-level technical services used to execute concrete actions; task components, that define abilities to manage the execution of higher-level activities, that are intended to become part of mission compositions; and an agent component, that contains a software agent that manages the component-based architecture of the embedded software.

An agent [4] is an active entity that is designed to collect information, reason, make decisions and act autonomously. We use agents' activity to endow devices with self-configuration. Such autonomic capabilities are essential to pervasive systems: manual intervention cannot cope with neither large scale or continuously changing systems. Moreover, an agent is a social entity that is designed to interact with other agents in order to manage complex activities as distributed collaborations. A multi-agent system provides decentralized, peer-to-peer, communication schemes that are naturally suitable for open and dynamic architectures of pervasive systems. Agents senses automatically the presence of other agents thanks to discovery protocols integrated to middlewares. They dynamically create a community, share information about the resources they control and plan the distributed execution of missions.

### *B. The Hydroguard Pervasive Water Surveillance System*

An application of our work is the control of the Hydroguard Pervasive Water Surveillance System[1]. This system is designed to monitor hydrological parameters on rivers and coastal areas in order to detect critical situations such as floods or pollutions. It is composed from on-site devices that use various types of sensors to measure meaningful physical quantities (pressure, temperature, rainfall, pH, etc.). Each site has a specific geographical situation. Its streams and waters have specific characteristics. Devices have to operate in various contexts, for which their software must be specifically configured, thanks to the chosen component-based approach. Moreover, this context is not fixed once and for all. As devices operate outdoor, they may endure bad weather conditions that cause communication losses or device failures. In emergency cases, these changes may be intentional: extra devices may be added to extend the monitored area or to get additional measures; conversely, devices may be moved to a more demanding site or to mitigate failures. Hydroguard is thus a pervasive system that must support dynamic arrival or leaving of devices. Its devices must react to these changes and adapt their behaviors and collaborations to maintain the continuity of their monitoring missions. This is handled by the autonomic dynamic adaptation capabilities provided by the embedded agents. This way, supervision by human operators is not required and the system may stay operational even when remote administration is impossible in exceptional situations.

Besides, as aforementioned, the system is designed to run multiple missions in parallel, in order to monitor multiple hazards. Depending on environmental situations, defined by domain experts in terms of thresholds on specific monitored data, the criticality of missions may evolve from normal (routine) to vigilance and finally to crisis. This may require to change the operation mode of some missions, for instance to timely follow the evolution of an incident, granting them higher priority so that the system concentrates its resources on the more critical and relevant missions first. Combined with the pervasive nature of the system, the presence or the availability of a resource is never guaranteed. Mission definitions must therefore include various admissible ways to achieve them, depending on criticality and resource availability, that agents use to find the best execution configuration, in any changing context. Mission definition is thus a key issue for system adaptation, extending the perimeter of

[1]Partly funded by the French government FUI-AAP8 project.

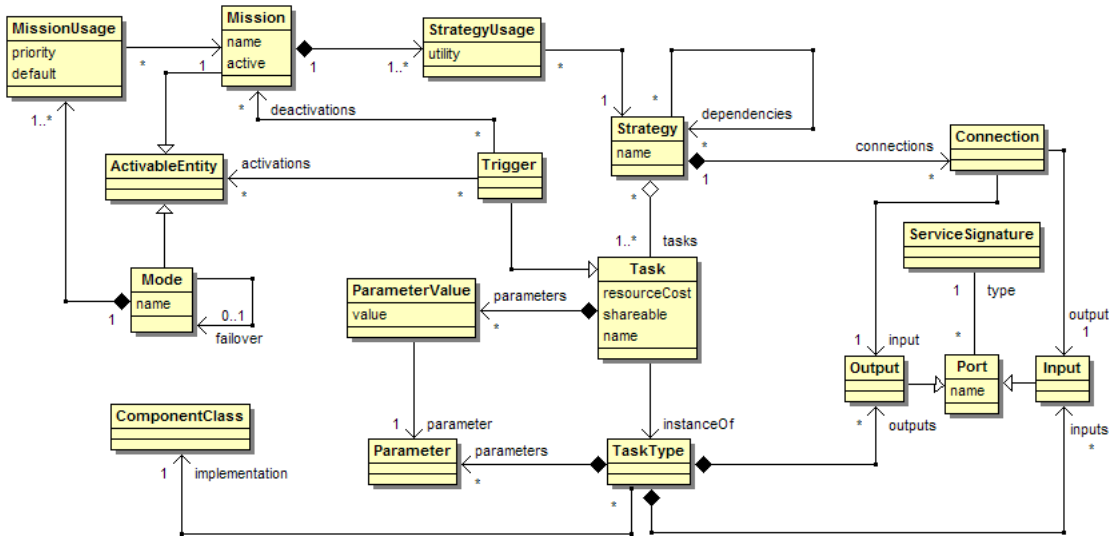


Figure 2. Metamodel of Arold, a multi-context mission description language

context-awareness [5] to user requirements. We address this issue with the proposal of Arold, a multi-context mission definition language.

## III. Arold: a multi-context mission description language

As discussed in the previous sections, pervasive systems have open, dynamic architectures that may evolve at any moment, in unanticipated ways. The invariant part of the system is the set of functional objectives it must achieve to meet the requirements of its users. Arold is proposed as a mean to capture users' requirements as abstract, declarative mission definitions. Missions definitions are then automatically mapped by the manager agents controlling the pervasive system to available resources so as to find an appropriate configuration to execute the missions in the current context. This way, missions are not defined by components that may disappear when devices leave the system. The main concepts of Arold, as shown in its metamodel (*see* Figure 2) are *modes*, *missions*, *strategies* and *tasks*.

### A. Modes and Missions

A *mission* defines a functional objective that is assigned to the system. Being open, one inherent characteristics of pervasive systems is to be composed from various devices and to support versatile missions. In Hydroguard, missions are, among many others, to forecast weather, to monitor flood level, to send flood alerts and to monitor water quality. Though every mission corresponds to some user requirements, all the missions do not have the same criticality and relevance, regarding specific situations of the environment. *Modes* thus model the different situations that the system is expected to manage distinctively. In Hydroguard, *Routine*, *FloodVigilance* and *PollutionCrisis* are mode examples. Modes build a first level of context representation, which defines the specific environmental situations that the system has to manage. Missions build a second level of context representation, which captures user requirements. Each mode explicitly specifies the set of missions that is to be achieved in the current situation (*see* Figure 3). Missions are defined as independent concepts, so they can be reused in as many modes as necessary. Mission criticality is expressed by a priority: *Mandatory*, *High*, *Normal*, *Low*. These priorities are affected by each mode, so that mission criticality may vary to be adapted to the context. Mandatory missions corresponds to critical activities. If any mandatory mission cannot be executed, the system cannot properly manage the current mode. It must signal a failure and enter a failover mode. Monitoring the conditions that determine transitions to other modes is part of the mandatory missions that must be associated to a mode. The system then tries to execute as many missions as possible, depending on available resources, starting by high priority missions and finishing with low priority missions. To enable a more flexible definition of modes, all the missions associated with a mode are not intended to always execute at the same time, but possibly successively, as the situation evolves. To manage this, missions hold a status attribute that specify if they are active or not. Apart from its mandatory missions, which are always active, a mode defines a set of default missions that are active when entering in this mode. The set of active missions then evolves by explicit mission activations or deactivations controlled by the already active missions.

### B. Strategies and Tasks

A *strategy* defines a concrete implementation of a mission. A mission can be implemented by a set of strategies (*see* Figure 3), that represent alternative ways to achieve the functional objective represented by the mission. Strategies build a third level of context representation, which defines how the execution of missions may vary to enable its adaptation to available resources. Strategies need not be strictly equivalent. On the contrary, the different strategies associated with a mission should represent various tradeoffs between resource requirements and utility. This way, among the strategies that can be currently executed, the system tries to find an optimal set of strategies, that is the set of strategies that enables to execute the largest set of active missions, with the best efficiency. The efficiency of a strategy depends on its cost (resource requirements) but most importantly to its utility, regarding the achievement of the mission. Utility is a score defined for each strategy by a mission, which characterizes the quality of the result that is produced by the execution of the strategy. Utility is contextual. Associated to a mission used in a routine mode, an economic strategy with a poor quality of service would be ranked with a low utility. Associated with a mission used in an emergency mode, it would be anyway useful and ranked with a high utility.

A strategy is concretely defined by a set of *tasks* that must be instantiated and connected together by the system in order to execute the mission (*see* Figure 3). This part of AROLD is thus equivalent to a simple architecture description language [6]. Each task represents a specific service invocation, specified by a set of parameter values. A task (*e.g.*, *WaterLevelCollector1*) corresponds to an instance of a task type (*e.g.*, *WaterLevelCollector*), so that different tasks can be defined corresponding to a common kind of activity. A task type defines the list of parameters (*e.g.*, *sensorIP, rate, levelID*) to be specified in each individual task. It also defines how the task type is implemented, as a reference to a concrete component (*e.g.*, *GenericWaterLevelSensor*) that must be used to instantiate this kind of tasks. Finally, a task type specifies a set of in and out ports (*e.g.*, the *WaterLevelCollector* task type has an out port as it is a data sink). In and out ports define the information (data, control) that a task respectively receives from or sends to others tasks. Task compositions are defined in strategies by sets of connections between in and out ports of their component tasks (*e.g.*, connection between the *WaterLevelDatasource1* task and the *WaterLevelCollector1* task). This can be regarded as similar to workflows. However, task compositions do not hold any explicit control structures in AROLD. Composition is thus purely structural and amounts to assemblies, as proposed in SCA [2]. Workflow control is embedded in tasks. When interactions are required, external control management is defined by ports. Ports are typed by service signatures (*e.g.*, the *WaterLevelDataSink* service saves a water level value along with a localization ID and a timestamp). These syntactical specifications are used to check the soundness of connections between ports.

```
mode FloodAlert
{
  active mission SendFloodAlarms priority high
  active mission CollectFloodData priority normal
  active mission MonitorFloodAlert priority mandatory
}
mission CollectFloodData
{
  strategy CombinedFloodDataCollection utility 100
  strategy SingleFloodDataCollection utility 75
}
strategy SingleFloodDataCollection
{
  task WaterLevelCollector1
  task WaterLevelDatasource1
  connection WaterLevelCollector1.data=>WaterLevelDatasource1.store
}
strategy CombinedFloodDataCollection requires SingleFloodDataCollection
{
  task PressureCollector1
  task PressDatasource1
  connection PressureCollector1.data=>PressDatasource1.store
}
task WaterLevelCollector1
{
  type WaterLevelCollector
  resourceCost 10
  parameter sensorIP=192.168.1.30:100
  parameter rate=5mn
  parameter levelID=LeftBankRiverSite1
}
taskType WaterLevelCollector
{
implementation hydroguard.sensor.water.GenericWaterLevelSensor
parameter sensorIP : String
parameter rate : String
parameter levelID : String
out port data : WaterLevelDataSink
}
service WaterLevelDataSink
{
saveWaterLevel(levelID:String,timeStamp:Date,value:float)
}
```

Figure 3. Excerpt of a mission descriptor in AROLD textual syntax

Finally, each task is defined by its resource consumption. For the sake of simplicity, it is represented in this paper as a unique attribute (*resourceCost*). In our concrete implementation, resource consumption encompasses CPU, RAM, disk and network usage. These attributes are first used individually to check that the instantiation of a task on a device does not exceed its amount of resources.

## IV. Optimized Mission Deployment

Taking into account an operation context modelled in AROLD, the role of the manager agents embedded in the devices composing the pervasive system is to determine an optimized deployment of the set of active

missions in the current active mode. As aforementioned, this deployment should be optimized so that, taking into account resource limitation, the executed missions are the most critical and the more useful to users. To calculate this optimal deployment is an optimization problem, with a high computational complexity, due to the choice of the missions, strategies and devices where tasks are instantiated. On a formal ground, the optimization of mission deployment is a constraint satisfaction problem analogous to a variant of the knapsack problem, which is a NP-hard problem [7]. This problem consists in filling up a knapsack with items of various volumes and values. The best combination of items must be chosen so that the knapsack the volume of which is fixed contains a maximal total value. In our mission deployment problem, the best combination of tasks must be allocated on the devices of the system, so that the total utility of the corresponding strategies, for the active missions in the current mode, is maximal. Mission deployment is calculated separately for each level of priority, so that resources are used by higher priority mission first.

For the sake of efficiency, we have implemented the resolution of the mission deployment problem as a centralized mechanism. When started, a manager agent queries the service repository of the OSGi platform to retrieve the list of the task components that are installed on the device it controls. It then broadcast a message to request the election of a leader among the group of manager agents of the pervasive system. We have designed an election protocol, derived from the bully algorithm [8], but adapted to the open structure of pervasive systems (the description of which is out of the scope of this paper). Centralization is thus mitigated because it is a dynamic process in which any manager agent can become the leader. When the manager agent receives the address of the elected leader, it sends back to the leader a list of its features, meaning its resource capacity and the list of the task types it can instantiate. It then waits for the leader to send configuration instructions, describing which tasks to instantiate and which connection to set up. Meanwhile, the leader collects the feature informations sent by the other manager agents. Taking into account the missions specified by Arold descriptors, these informations enable the leader to build a global definition of the optimization problem to be solved. This is a point where centralization becomes an advantage: the more complete is the problem definition, the more optimized the found solution may be.

Many resolution algorithms exist for the knapsack problem. Our application is the dynamic reconfiguration of a pervasive system. The optimization problem must therefore be solved not only timely but also on a device with limited resources. We thus designed an adaptation of the greedy approximation algorithm, chosen for its very low time and space complexity. The strategies associated with active missions are ordered by their efficiency (ratio between their utility and resource consumption). The possible deployment of the most efficient strategies are checked first (most rational choices). Strategies are thus chosen, until every active mission is executed by a strategy or no more strategy can be deployed because of resource unavailability.

Determining where a task should be allocated, between a set of candidate devices, is another source of combinatorial explosion. To solve this issue and preserve the linear complexity of the greedy algorithm, we designed a choice algorithm using an ordered list of devices, similar to the ordered list of strategies. Based on both the set of tasks that appear in the mission descriptors and the list of features transmitted by each manager agent, a probability that a task has to be affected on a device is calculated. A potential load is calculated for each device. A task is allocated on the device that has the lowest potential load: this is here again the most rational choice to avoid resource shortage and to tend towards a balanced load between devices. After each allocation, probabilities and potential loads are updated, to take into account the already known task allocations. When the allocation of a task is impossible, the corresponding strategy is skipped. Deployment resolution carries on with the next more efficient strategy to be tried. This way, deployment is solved with a linear complexity that fits the computation capabilities embedded in devices. Though simple, this algorithm is able to produce on average interesting suboptimal results.

## V. Related Work

Arold deals with service compositions, a much studied topic related to web services choreographies but also pervasive systems [1], [9], [10], [11]. These works focuses on a transparent management of the execution context, to let users declaratively express their required compositions, without any specific knowledge about the environment. The supporting frameworks deal with a wide range of mechanisms: discovering available services, matching available services with user defined compositions, maintaining execution by automatically replacing faulty services, balancing execution load. Goal-oriented systems even go one step further. Users express the results they wish to obtain and the system automatically calculates, by backward-chained inferences, theservice compositions that achieve the specified results [12]. All these systems consider the context from a technical point of view, that can thus become transparent for the user. With Arold, user expertise and preferences are part of the context definition, modelling variability and utility as alternate strategies. The primary goal of the system becomes to best satisfy all its users: the aforementioned

problems are then considered as constraints in a decision problem (mission and strategy selection).

Multi-agent systems are often proposed to manage pervasive systems [13], [14], [15], [16], [17], as they both have open and dynamic architectures. Autonomic adaptation is handled in a distributed way and emerges as the result of peer-to-peer negotiation protocols. On the opposite, we have chosen a centralized mechanism for mission deployment , adapted from the optimized execution plans used in grid-computing [18]. Though very simple, to suit the limited resources of the devices, we consider that it is able to produce faster more optimized results than the convergence of a distributed process.

## VI. Conclusion and future work

This paper presented a work in progress. Arold is an original contribution, as a service composition language that enables the contextual adaptations of the missions of a pervasive system to be defined. It takes into account mission criticality, resource availability and user preferences.

Dynamic context redefinition and multiple context definition mitigation still are open issues. Future work will have to detect and manage inconsistencies. Task mobility will also be studied to support the redeployment of running missions and its cost evaluated. Experiments still have to be run to validate the performance hypothesis about our proposed centralized mission deployment scheme as compared to a massively distributed alternative. To do so, we plan to run simulations of large scale distributed arcitectures on a machine cluster.

## Ackowledgments

The authors would like to thank the participants of the Hydroguard consortium.

## References

[1] Bronsted J., Hansen K. M., and Ingstrup M.: Service Composition Issues in Pervasive Computing. IEEE Pervasive Computing, vol. 325, pp. 311-326 (2011).

[2] Open SOA Collaboration: Service Component Architecture Specification. http://www.osoa.org/, accessed 1-july-2012, (2006).

[3] OSGi Alliance: Osgi service platform Release 4.http://www.osgi.org, accessed 1-july-2012, (2005)

[4] Wooldridge M.: An Introduction to Multi-agent Systems. Wiley (2002).

[5] Bolchini C., Curino C. A., Quintarelli E., Scheiber F.A., and Tanca L: A data-oriented survey of context models. ACM SIGMOD Record, vol. 36(4), pp.19-26 (2007).

[6] Medvidovic N. and Taylor R. N.: A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering, vol. 26(1), pp. 70-93 (2000).

[7] Kellerer H., Pferschy U., and Pisinger D.: Knapsack Problems. Springer (2004).

[8] Mamun Q.E.K., Masum S.M., and Mustafa M.A.R.: Modified bully algorithm for electing coordinator in distributed systems. WEAS Transactions on Computers, vol. 3(4), pp.948-953 (2004).

[9] Ibrahim N. and Le Mouel F.: A Survey on Service Composition Middleware in Pervasive Environments. International Journal of Computer Science Issues, Vol. 1, pp. 1-12 (2009).

[10] Urbieta A., Barrutieta G., Parra J., and Uribarren A.: A survey of dynamic service composition approaches for ambient systems. Proceedings of the Ambi-Sys workshop on Software Organisation and MonIToring of Ambient Systems, ICST, pp. 1-8 (2008).

[11] Bakhouya M. and Gaber J.: Service Composition Approaches for Ubiquitous and Pervasive Computing Environments: A Survey. Agent Systems in Electronic Business, IGI Publishing, pp. 323-350 (2007).

[12] Heider T. and Kirste T.: Multimodal appliance cooperation based on explicit goals : concepts and potentials. Proceedings of the Joint Conference on Smart objects and Ambient Intelligence, ACM, pp. 271-276 (2005).

[13] Malek S., Mikic-rakic M., and Medvidovic N.: A Decentralized Redeployment Algorithm for Improving the Availability of Distributed Systems. Proceedings of the 3rd International Conference on Component Deployment, Springer, pp. 99-114 (2005).

[14] Weyns D., Malek S., and Andersson J.: On Decentralized Self-Adaptation: Lessons from the Trenches and Challenges for the Future. Proceedings of the ICSE workshop on Software Engineering for Adaptive and Self-Managing Systems, ACM, pp. 84-93 (2010).

[15] Bakhouya M. and Gaber J. : Self-organizing Approach for Emergent Multi-agent Structures. Proceedings of the Workshop on Complexity through Development and Self-Organizing Representations, ACM (2006).

[16] Grondin G., Bouraqadi N., and Vercouter. L.: MADCAR, an Abstract Model for Dynamic and Automatic (Re-)Assembling of Component-Based Applications. Proceedings of the 9th International SIGSOFT Symposium on Component-Based Software Engineering, LNCS 4063, Springer, pp. 360-367 (2006).

[17] Hamoui F., Huchard M., Urtado C., and Vauttier S.: SAASHA: a Self-Adaptable Agent System for Home Automation. Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE, pp. 227-230, (2010).

[18] Parashar M. and Pierson J. M.: Pervasive Grids: Challenges and Opportunities. In Handbook of Research on Scalable Computing Technologies, IGI Global, pp. 14-30 (2010).