# PROTEUS: A Language for Adaptation Plans

Antinisca Di Marco, Francesco Gallo
*University of L'Aquila*
*Department of Information Science*
*L'Aquila, Italy*
*antinisca.dimarco, francesco.gallo@univaq.it*

Franco Raimondi
*School of Engineering and Information Sciences*
*Middlesex University*
*London, UK*
*f.raimondi@mdx.ac.uk*

***Abstract*—The purpose of this paper is to present PROTEUS, a new language and, more in general, an approach for the construction of reconfiguration plans to support adaptation in systems belonging to different domains. The approach allows the management of runtime adaptation, preventing that running shared services are terminated and taken off-line while being reconfigured, causing inefficiency and disruptions. We introduce the new concept of *virtual membrane*, in order to give the system ability to adapt itself at run time in front of a new reconfiguration plans. We apply PROTEUS on an example to show the expressiveness and power of the new language.**

***Keywords-Adaptive Systems. Reconfigurations Rule. DSL. ADL.***

## I. Introduction

Modern software systems show a complexity and a size that can often make difficult their maintenance and adaptation to environmental changes. Moreover, there is an increasing need to satisfy functional and non-functional requirements when a system lives in an environment composed by independent and often competing entities, such as in the web service market [1][2]. The current trend is to delegate adaptation [3][4] and fault tolerance [5] to the system itself by means of redundancy and other techniques.

To support these capabilities, we propose a new reconfiguration language called PROTEUS, that aims at building and managing rules to reconfigure software applications. The new rules are generated and managed using a new concept: the Virtual Membrane. A virtual membrane defines one input for a new reconfiguration plan in which all the resources involved are subject to adaptation. The intuition is that rules represent views of a certain system and define a "new" configuration for the system which is compatible with the new state of the environment (i.e., the new context in which the system lives). Note that it is out of scope of this paper to define how the reconfiguration plans are created. For this aim, we envisage an engine (local to the system) able to interpret the context changes and to define the suitable reconfiguration plan (expresses by means of PROTEUS) to adapt the system to the new situation. In our mind, such engine needs to be distributed across the system modules/resources. The rest of the paper is organized as follows: Section II describes PROTEUS and fundamental concepts related to it, using a scenario that describes the management of a WSN (Wireless Sensor Network) for the detection of a traffic jam. In Section III and IV, we formally introduce the main concepts of PROTEUS. Related work is presented in Section V. Section VI provides concluding remarks and future work.

## II. Motivational Example

In this section, we introduce the Traffic Jam case study to which we apply some of the most significant actions of reconfiguration introduced by PROTEUS and the concept of virtual membrane. With this example, we want to show the ability of PROTEUS to manage the system resources at a high-level, and to introduce the concept of virtual membranes, which allows to separate the behavior from the system resources and organize them so as to have different independent views.

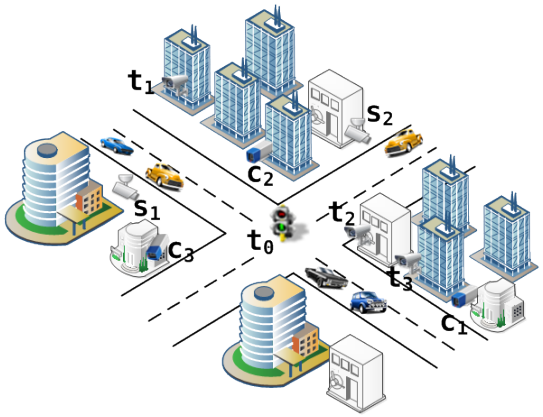


Figure 1. Scenario

The scenario of Figure 1 shows an urban environment, consisting of roads, buildings, cars and a traffic jam control system composing of a series of sensor nodes installed along the ways or on buildings. The system is composed of a set of wireless nodes of various types and one base station contained in the traffic light. The role of base stations is created at run-time by means of clustering algorithms [6], capable to elect a cluster head. In case of failure of the base

 ISBN: 978-1-61208-219-6 

station, a new cluster head(base station) is elected among the nodes with sufficient computing power. As we will see in Section III-B, we identify the various components of the system with a set of *resources*, namely $RES$. For the sake of this example, $RES$ is defined as follows:

$$RES = \{t_0, t_1, t_2, t_3, c_1, c_2, c_3, s_1, s_2\}$$

where:

- $t_0$ represents the base station/traffic light; it receives data from various camera nodes and turns on the traffic lights if necessary. It also has a greater computing power than other nodes on the network;
- $t_i$ are cameras having the ability to rotate around their vertical axis. They can not change their angle of inclination and are equipped with temperature sensors and sensors to detect $CO_2$ levels;
- $c_j$ are cameras having the ability to rotate around their vertical axis and to vary their angle of inclination.
- $s_k$ are fixed cameras that can cover only a limited part of the road.

We assume that all nodes are powered by a solar panel, can communicate with their neighbour nodes, and the base station has two types of clients:

1) the system for the management of the traffic light that is activated when a traffic jam is detected;
2) the system for detection of $CO_2$ level. An alert is triggered when this level exceeds a given limit value of $CO_2$.

As mentioned above, only the nodes of type $t_i$ have the capacity to measure the level of $CO_2$. We call this feature $f_1$, and we extend the set of resources $RES$ with this additional feature:

$$RES = \{t_0, t_1, t_2, t_3, c_1, c_2, c_3, s_1, s_2, f_1\}$$

The two base station clients have a different logic view of the network, since they have different monitoring purposes. In our approach these two logic views can be synthesized by two *virtual membranes*:

$$v_1 = \{t_0, t_1, t_2, t_3, c_1, c_2, c_3, s_1, s_2\}$$

and

$$v_2 = \{t_0, t_1, t_2, t_3, f_1\}$$

In our formalism, virtual membranes are members of the set of resources, and thus

$$RES = \{t_0, t_1, t_2, t_3, c_1, c_2, c_3, s_1, s_2, f_1, v_1, v_2\}$$

Consider now the following scenario: due to a technical problem, some of the nodes belonging to the membrane $v_1$ start sending incorrect data to the base station $t_0$, causing improper behaviour of the traffic lights.

At this point we need to ensure that the base station does not continue to receive incorrect data from sensors in $v_1$, still guaranteeing the management of the traffic light in case a traffic jam is detected.

It is also necessary to select in an appropriate way the system resources that will be affected by the reconfiguration plan. We do this by defining a predicate $P_1$ defined as follows:

- $P_1 = \{r_i == v_1\}$

Intuitively, the predicate is true for a resource iff that resource is the virtual membrane $v_1$. We employ this predicate in PROTEUS to send a reconfiguration message that deactivates the $v_1$ virtual membrane; that is, it makes inactive the corresponding system view or behavior. The syntax for this reconfiguration plan is as follows:

$\{\forall\, r_i \in RES :\ P_1(r_i) = true)\} \Longrightarrow$
$program\,\{\ \ vInactive(self, vMembrane\, v_1);$
$\}$

We refer to [7] for the definition of the full syntax of PROTEUS. We want to recall by turning off $v_1$, the behavior implemented by it is inhibited. Whereas, inactivating $v_1$ does not inactivate or stop the resources belonging to it. This means that, if the resources used by $v_1$ are shared with other virtual membranes, the manipulation of $v_1$ is absolutely transparent to them since the shared resources continue to work for them. In our example, this means that all the resources in $v_2$ shared with $v_1$ continue to work even if $v_1$ is inactive.

Once the virtual membrane $v_1$ is inactive, to continue to provide the traffic light management in case a traffic jam occurs, we use membrane $v_2$: the excessive production of $CO_2$ in a particular point of a road network presumes a large number of combustion engines in transit, or a high density of vehicles not in motion, hence a traffic jam.

We can then use $f_1$ to enable the semaphore when the $CO_2$ level exceeds a certain threshold by means of the following property and reconfiguration plan:

- $P_2 = \{r_i == f_1\}$

$\{\forall\, r_i \in RES :\ P_2(r_i) = true)\} \Longrightarrow$
$program\,\{$
$vCreate\, v_3\{$
$v_3(P_2)\{$
$if(f_1 == threshold)\{active(self, t_0)$
$\}$
$if(f_1 < threshold)\{deactive(self, t_0)$
$\}$
$\}$

In this reconfiguration plan, a new virtual membrane is created (namely $v_3$) that is composed only by $f_1$. Note that the reconfiguration plan makes use of $f_1$ to turn on or off the traffic light depending on the $CO_2$ level. Indeed, $f_1$ is a feature implemented by the resources in $v_2$ hence $v_3$ uses implicitly the resources belonging to $v_2$. Even if $v_2$ and $v_3$ share implicitly and explicitly the same resources, they do not affect each other since the former is transparent to the latter and viceversa.

 ISBN: 978-1-61208-219-6 

## III. SETTING THE CONTEXT

In this section, we provide an abstract formalization of the concepts introduced in the previous section. An adaptive system is a system able to provide adaptation features, following the application of specific capabilities. In PROTEUS, an adaptive system is composed of two logical layers:

**Application Layer**, which represents the application logic of the system where all hardware and software features, and their relationships are defined and managed;

**Adaptation Layer**, providing features for reconfiguration and adaptation. In particular, this layer is able to process the reconfiguration specified by PROTEUS.

### A. Event Concept

The adaptation is triggered by external or internal events; once the system has captured one of these events, it creates a virtual membrane that selectively aggregates the resources necessary to implement the plan of reconfiguration.

Formally, an event is a logical predicate, which can have the following forms:

$$\{\forall\, r_i \in RES:\ P(r_i) = true)\} \Longrightarrow reconf$$

or

$$\{\exists\, r_i \in RES:\ P(r_i) = true\} \Longrightarrow reconf$$

where P is a property from a $properties\_set$, which determines the resource set to select; $reconf$ is the new behaviour to be implemented. At the present, we assume that the reconfiguration plan is generated manually or by an external entity.

### B. Resource Concept

In PROTEUS, the concept of resource plays a crucial role; it is an aggregation of *features* and *attributes*, as described in Figure 2. Both features and attributes are the resources on which it is possible to act directly.

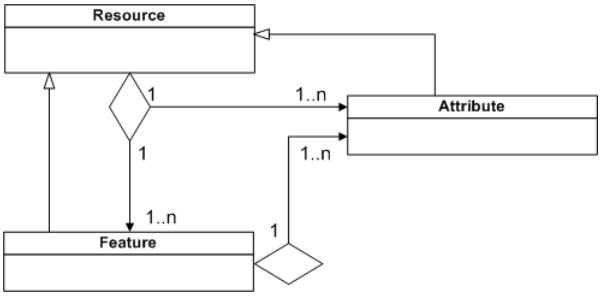


Figure 2. Resource concept

From a formal point of view, resources are assimilable to elements that belong to a *multiset* (the concept of *multiset* is a generalization of the concept of set where the multiplicity of any element could be greater than one.): this allows us to group together instances of various types (and possibly the same type), and act on them in a distributed mode. More formally, we identify resources with the set $RES$, where:

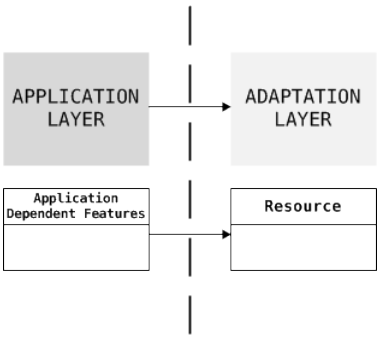


Figure 3. Resource and Feature

**Definition:** $RES = \{r_1, r_2, ..., r_n, vm_1, ..., vm_k\}$ is a *multiset* in which all the resources that are part of the application layer are collected. Each of these resources has a *status* that tells the user if it is *active* or *not active*. The multiset can contain both hardware and software resources, but also logical features represented by virtual membranes.

Whenever the system is involved in an adaptation, PROTEUS is able to add or remove resources, enable or disable existing features, etc..

Thanks to the concept of attribute, PROTEUS is able to further enhance the capabilities of a resource, allowing the user to change the state of the resource or activate components of the resource made temporarily silent, or added later (where possible). In this sense, we have:

**application-depended features**, that specialize the generic concept of feature resources, depending on the application domain and;

**adaptation-depended features**, that express the logic to adapt the features of the application layer. Figure 3 describes the concept of abstraction that PROTEUS introduces: it allows us to keep separate the application logic of the system from the logic of adaptation, ensuring the following advantages:

- the *adaptation layer* is generic, i.e., it is not bound in any way to the application layer. In this way the complexity of the application layer of the system is not weighed down by specific reconfiguration logic;
- the *application layer* can be customized according to the domain application, to support specific needs of adaptation.

### C. Resource Type

Each resource has a *type* that characterizes and distinguishes it in the system. Our approach provides two kinds of *resource types*:

**Adaptation Related** resource types are the mechanism through which our approach allows to insert new behaviors

in an application, or update a resource or collection of resources. This corresponds, for example, to adding a method $m_j$ to a class $C_i$ or to deactivating a feature, or a part of it.

**Application Related** resource types are determined by the application domain. For example, consider the WSN domain described in Section II: an application type may be defined by a particular sensor, called $Sensor_j$.

## IV. PROTEUS MAIN FEATURES

In this section, we present the main features of PROTEUS such as the adaptation actions (in Section IV-A), properties (in Section IV-B) virtual membrane (in Section IV-C) and the actions PROTEUS defines for it (in Section IV-D). The interested reader can refer to [7] for the complete syntax of PROTEUS.

### A. Adaptation Action

In this section, we describe the principal actions that PROTEUS provides.

In Proteus a reconfiguration plans is a sequence of *adaptation action*s as summarized in Table I. These actions are described as follows:

- *add* allows the user to add some resource to the system. We recall that a resource can be: *vMembrane*, *feature*, *attribute* or *domain specific*. For example, if the application domain is a wireless sensor network, we may want to add to the network a new type of sensor, called $Sensor_j$;
- *bind* between resources. For example, considering a WSN, a bind may reflect the need to initiate a communication between a base station and a peripheral sensor;
- *remove* resources. For example, considering a WSN, remove a resource could mean that a node is unavaible due to failure;
- *update* resources. For example, this allows the user to update an old version of a software component, change the communication protocol between nodes on a network, etc.;
- *activate* resources. For example, it allows to activate one particular sensor of the WSN nodes;
- *deactivate* resources. For example, if a network of sensors needs to reduce energy consumption, the user can disable one or more of the sensing node modules.
- $domain\_specific$ defines domain specific adaptation actions the application developer wants to introduces in the language. For example, a developer could build reconfiguration actions that are a combination of those above in order to make atomic a sequence of them or reconfigure a WSN according to a certain policy.

### B. Properties

For each resource identified, we associate a set of constraints required to change the current configuration of the system or subsystem. We can simultaneously select a set of resources to apply an adaptation plan consisting of a sequence of actions. The resources spaces (features) involved are selected through the use of *properties*, formally defined as:

***Properties:*** $PRO_{j=\{1..m\}} = \{constraints^{+}\}$, is the set of properties that the system must satisfy. Each property is defined by one or more constraints, represented by logical predicates.

### C. Virtual Membrane

In order to adapt a single resource or a pool of resources, we introduce, through PROTEUS, the concept of **Virtual Membrane** that provide tools that support the ability to adapt the system to internal or external events. A virtual membrane defines the boundaries of the portion of the application subject to adaptation. The purpose of a membrane is to select a resource or group of resources belonging to the system, to define new interactions within the system and, consequently, new behaviors, or to modify the behavior of the internal resources of the application.

The advantages of introducing the concept of virtual membranes are the following:

- the adaptation operations are performed at runtime, ensuring continuity of service;
- the creation of a virtual membranes allows the selection of a set of resources, generating a specific view of (the portion of) the system involved in the change. This view does not interfere with the various other views of the system since they are *behaviors*;
- the previous point has the consequence that the adaptation is completely transparent to the user. Each view gives a level of abstraction that is only accessible to the user/subsystem which are enabled, Figure 4;

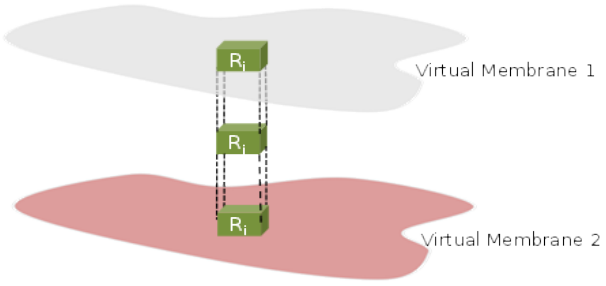


Figure 4. Virtual Membrane

- once created, the virtual membranes are comparable to system resources; in fact they can be manipulated through specific actions.

**Definition:**

***Virtual Membrane:*** A virtual membrane $vm_j = \{r_i \in RES : PRO_j(r_i) = true, i = \{1..n\})\}_{j=\{1..m\}}$ is a set that contains all elements of the system that verify the property $PRO_j$.

| Adaptation Action | Description |
|---|---|
| **Add** | The action allows the user to add some element to the system. The element type can be: *vMembrane*, *feature*, *attribute* or *domain specific*. |
| **Bind** | The action creates a bind between two system's resources. |
| **Remove** | The action allows the user to remove a resource from the system. The removal of a resource is final and it can not be used any longer. |
| **Update** | The action updates a system resource. |
| **Activate** | This action allows the user to enable existing resource. |
| **Deactivate** | This action allows the user to disable temporarily or permanently a resource. |
| **+domain_specific** | Developers can define their own action, depending on particular applications needs. |

Table I
ADAPTATION ACTION

***Virtual Membrane Sets:*** $VM = \{vm_j\}_{j=\{1..m\}}$ is the set of all virtual membranes generated in the system.

In PROTEUS virtual membranes are considered as standard resources: this is to be able to adapt the virtual membrane in a uniform way with others concepts. In this way, the virtual membrane is itself adaptable and it can evolve over time. To manage this type of resource we defined specific actions, called *Virtual Actions*, specified in the following section.

### D. Virtual Action

PROTEUS specifies a set of actions to create and manipulate virtual membranes that are listed in Table 2. Recall that a virtual membrane is a logical resource of the system, and thus the actions reported affect the behavior associated with the virtual membrane, rather than the resources that implements it.

## V. RELATED WORK

In this section, we summarize the characteristics of some architectural styles and frameworks useful to develop and manage reconfiguration in various types of software systems.

We discuss the state of the art by comparing it with PROTEUS. To this end we organize the presentation of the main features of the related work in Table III. Each row of the table is dedicated to an approach whereas the first row is related to PROTEUS and provides the reconfiguration actions of PROTEUS that we consider as the minimum set for the construction of a language able to define a reconfiguration plan.

The approaches described here are applicable to levels of granularity ranging from the reconfiguration of the architectural elements to objects in the actual running the system.

The various tools considered are as follows:

- **FScript** [8], a Domain-Specific Language used for reconfigurations of Fractal architectures.
  FScript introduces a new notation, called FPath, which is designed to express queries on a FRACTAL architecture and navigate it, selecting items on the basis of logical predicates. FScript provides access to all of the primitive actions present in Fractal reconfiguration, and enables the user to define customized reconfiguration.
- **FORMAware** [9], incorporates component-based development and architecture knowledge. Furthermore, this framework provides flexible mechanisms to recompose components at runtime to address scalability, mobility and general architecture evolutionary scenarios.
- **Dynamic Contextual Adaptation with a DSL** [10], defines high-level declarative constructs that can be used to specify the adaptation of the application behaviour to specific situations. The language is supported by a framework that enables the exchange and merge of behaviours on-the-fly.
- **DSL for ATRON robots** [11], is a role-based language that allows the programmer to define roles and behaviour for a physical module that is activated when the structural invariants associated with the role are fulfilled.
- **Representational state transfer (REST)** [12], is a paradigm for Web applications that allows the manipulation of resources using methods GET, POST, PUT and DELETE of the HTTP protocol. Basing its foundations on the HTTP protocol, the REST narrows its paradigm field of interest to applications that use this protocol to communicate with other systems.
- **CLOUD Computing** [13], refers to a collection of technologies that allow store/archive and/or data processing (via the CPU or software) typically in the form of a service offered by a provider to the customer, through the use of hardware/software on a distributed and virtualized network.

In Table III, we show the reconfiguration actions that the various frameworks provide. For simplicity, the names of the actions of reference used are those introduced by PROTEUS: in each corresponding cell there is a brief description of the semantics of the action in reference to the particular framework.

With regard to REST, the adaptation can be conceived as the ability of the system (web) to expose applications and features, such as services (web), in the form of callable API from a client. Through the use of various kinds of connectors we can define a large number of interactions between clients and resources, facilitating the system scalability and

 ISBN: 978-1-61208-219-6 

| Virtual Action | Description |
|---|---|
| **vCreate** | This action allows to **build** the membrane by means of a constructor that is capable of aggregating elements of the systems that satisfy the properties introduced by the event that triggered the adaptation. |
| **vCompose** | This action allows to **combine** two virtual membrane through a *compose operation* set. The operations are *union*, *difference* and *intersection*. The use of these operators are allowed in their standard sense because the system resource set is defined as multiset. |
| **vRemove** | This action allows to **remove** from the system the constraints that generated the specific virtual membrane. |
| **vActive** | This action allows to **enable** the specific behaviour implemented a virtual membrane. This action affects the *status field*. |
| **vInactive** | This action allows to **disable** the specific behaviour implemented a virtual membrane. This action affects the *status field*. |

Table II
VIRTUAL ACTION

| | **Adaptive Action** | | | | | | |
|---|---|---|---|---|---|---|---|
| PROTEUS | Add | Bind | Remove | Update | Activated | Inactivated | Customization |
| FRACTAL | **ADL module**: - Composite components; - Instantiation a component from a ADL definition (Java code generation); - Shared components; - Content Controller (add sub component); - **Binding Controller**: - bind; | **Communication path between component interfaces**: - *Primitive Binding,* is a binding between one client interface and one server interface, in the same address space; - *Composite Binding,* is a communication path between an arbitrary number of component interfaces, of arbitrary language types; | **Content Controller**: - remove sub component; **Binding Controller**: - unbind; | | | | **FScript, FPath** |
| FORMAware | - Style Manager; - Architecture Graph; - Architecture Management; | - Style Manager; - Architecture Management; | - Style Manager; - Architecture Graph; - Architecture Management; | - Style Manager; - Architecture Management; | | | **ADL** |
| Dynamic Adaptation with a DSL | - Exchange Behaviour; - Merge Behaviour; | | | | Start Context | Stop Context | **DSL** |
| DSL for ATRON Robots | Role based | Role based | Role based | Role based | Role based | Role based | Role based |

Table III
ADAPTIVE ACTION IN RECONFIGURATION TOOLS

adaptability of the clients that use it. However, the following considerations apply:

- the client is bound by the number of exposed services;
- the client cannot act on the characteristics of resources, because these are completely transparent to the client.

CLOUD computing extends the concept of service performed in REST, adding two more levels: in the first, the services are identified by a platform (PaaS: Platform-as-a-Service), and services are identified by a set of programs or libraries. In the second, the services are provided by an entire hardware infrastructure (IaaS - Infrastructure as a Service). Even in the case of the CLOUD, the characteristics of resources are not visible to the client, which is just a user resource consumption, and it is obviously ”limited” by the available services.

From Table III, it is clear how the management of the reconfiguration of a system, depending on an external or internal event, is often delegated to the architectural level. It is therefore significant the use of Architecture Description Language (ADL)[14] for modelling the system. In addition to architectural languages, we considered Domain Specific Languages (DSL)[15], which can model a particular domain or a particular technical solution. The combination of these two technologies allows to:

- navigate in a selective way the elements constituting the software architecture and;
- operate dynamically reconfiguration operations on the elements that constitute the system.

Differently from all the approaches here surveyed, PROTEUS introduces the concept of Virtual membrane and the corresponding action (virtual actions) to manage it at run time as a usual resource. These aspects make PROTEUS innovative and powerful.

## VI. Conclusions and Future Work

In this article, we introduced PROTEUS. This language is characterized by the concept of virtual membranes, an abstraction that is designed to support the capacity of a system to meet the needs of an adaptation of its behavior, caused by an internal or external event.

Furthermore, we presented a simple application of PROTEUS, to show how the language can be used to implement reconfiguration plans. PROTEUS is still at an early stage of development, so it needs refinement and development. A formal semantics is currently being developed, in conjunction with a concrete implementation, to assess its ability to be used in different application contexts, its performance and its ability to scale in front of a large number of adaptations and its ease of use.

Concerning the implementation, we have already identified some tools to realize the concept of virtual membranes. In particular, we plan to use the SCALA language [16], and the concepts of traits [17] and class boxes [18].

## Acknowledgment

This work has been supported by the EU-funded VISION ERC project (ERC-240555).

## References

[1] H. T. Pu and Y. W. Wong, "User navigation behavior of a selective dissemination of web information service." in *iConference*, 2012, pp. 453–455.

[2] D. Vazhenin, "Cloud based web service for health 2.0." in *HCCE*, 2012, pp. 240–243.

[3] J. Dowling, T. Schäfer, V. Cahill, P. Haraszti, and B. Redmond, "Using reflection to support dynamic adaptation of system software: A case study driven evaluation," in *Proceedings of the 1st OOPSLA Workshop on Reflection and Software Engineering: Reflection and Software Engineering, Papers from OORaSE 1999*. London, UK, UK: Springer-Verlag, 2000, pp. 169–188. [Online]. Available: http://dl.acm.org/citation.cfm?id=646954.713478

[4] C. Ghezzi, M. Pradella, and G. Salvaneschi, "An evaluation of the adaptation capabilities in programming languages," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '11. New York, NY, USA: ACM, 2011, pp. 50–59. [Online]. Available: http://doi.acm.org/10.1145/1988008.1988016

[5] L. Sitanayah, K. N. Brown, and C. J. Sreenan, "Fault-tolerant relay deployment based on length-constrained connectivity and rerouting centrality in wireless sensor networks." in *EWSN*, 2012, pp. 115–130.

[6] E. Ever, R. Luchmun, L. Mostarda, A. Navarra, and P. Shah, "UHEED - an unequal clustering algorithm for wireless sensor networks." in *Sensornets 2012*, 2012.

[7] A. Di Marco, F. Gallo, and F. Raimondi, "Proteus language," http://www.slrtool.org/proteus/index.php/Proteus, University of L'Aquila, Tech. Rep., 2012. Last access 04/05/2012, University of L'Aquila, Tech. Rep., 2012.

[8] P.-C. David and T. Ledoux, "Safe dynamic reconfigurations of fractal architectures with fscript," in *Proc. Fractal CBSE Workshop, ECOOP'06*, 2006.

[9] R. S. Moreira, G. S. Blair, and E. Carrapatoso, "FORMAware: Framework of reflective components for managing architecture adaptation." in *SEM*, 2002, pp. 115–129.

[10] S. Fritsch, A. Senart, and S. Clarke, "Addressing dynamic contextual adaptation with a domain-specific language," in *Proceedings of the 29th International Conference on Software Engineering Workshops*, ser. ICSEW '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 188–. [Online]. Available: http://dx.doi.org/10.1109/ICSEW.2007.26

[11] U. Schultz, D. Christensen, and K. Stoy, "A domain-specific language for programming self-reconfigurable robots," APGES 2007, Automatic Program Generation for Embedded Systems, 2007.

[12] R. T. Fielding, "Architectural styles and the design of network-based software architectures," University of California, IRVINE - 2000.

[13] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, U.S Department of Commerce - Special Publication 800-145, 2011.

[14] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Trans. Softw. Eng.*, vol. 26, no. 1, pp. 70–93, Jan. 2000. [Online]. Available: http://dx.doi.org/10.1109/32.825767

[15] A. van Deursen, P. Klint, and J. Visser, "Domain-specific languages: an annotated bibliography," *SIGPLAN Not.*, vol. 35, no. 6, pp. 26–36, Jun. 2000. [Online]. Available: http://doi.acm.org/10.1145/352029.352035

[16] M. Odersky, "Scala language," online: http://www.scala-lang.org/.

[17] A. Bergel, S. Ducasse, O. Nierstrasz, and R. Wuyts, "Stateful traits and their formalization," Journal of Computer Languages, Systems and Structures, vol. 34, no. 2-3, 2008, pp. 83-108.

[18] S. Ducasse, "Supporting unanticipated changes with traits and classboxes," in *In Proceedings of Net.ObjectDays (NODE05*, 2005.