PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 32/09

A Self-adaptive Process that Incorporates a Self-test Activity

Andrew Diniz da Costa

Viviane Torres da Silva

Carlos José Pereira de Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 32/09 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena November, 2009

A Self-adaptive Process that Incorporates a Self-test Activity

Andrew Diniz da Costa, Viviane Torres da Silva[1],
Carlos José Pereira de Lucena

[1]Departamento de Informática – Universidade Federal Fluminense (UFF)

acosta@inf.puc-rio.br, viviane.silva@ic.uff.br, lucena@inf.puc-rio.br

Abstract. The self-adaptation paradigm aims to develop software systems that can autonomously adapt themselves to context changes and handle adverse situations on their own. However, appropriate implementation of self-adaptive processes or architectures able not only to check the needs for the adaptations and perform them but also to ensure their compliance with new environment requirements is still an open issue. Therefore, this paper proposes a self-adaptive process that contemplates a new activity that promotes the *test* of the adaptations in order to check their compliance with the new requirements. Our approach extends a basic self-adaptive process composed of four main activities (monitor, analyze, plan and execute) by including the *test* activity that will check the adapted behavior before its execution. The applicability of the proposed process is demonstrated by a case study where a system responsible for generating susceptibility maps, i.e., maps that show locations with landslides risks in a given area, uses the self-adaptive process to adapt its behavior and to check the adaptations before using them.

Keywords: Self-adaptation, self-testing, control loop, dynamic environment.

Resumo: O paradigma de auto-adaptação visa desenvolver aplicações que possam se auto-adaptar devido mudanças de contexto e para tratar situações adversas. No entanto, a implementação apropriada de processos ou arquiteturas de auto-adaptação deve considerar além da necessidade de realizar adaptações, a necessidade de garantir que tais mudanças estão sendo realizadas de forma apropriada, respeitando novos requisitos do ambiente. Assim, o artigo propõe um processo de auto-adaptação que oferece uma nova atividade que realiza testes em possíveis adaptações. Nossa abordagem estende um processo básico de auto-adaptação composto por quatro atividades principais (monitor, análise, plano e execução) incluindo a atividade de *teste* que possui a responsabilidade de checar o comportamento adaptado antes de sua execução. A aplicabilidade do processo proposto é demonstrado por um estudo de caso onde o sistema possui a responsabilidade de gerar mapas de suscetibilidade, isto é, mapas que indicam localizações com riscos de deslizamento em uma dada área e que usa o processo de auto-adaptação para adaptar seu comportamento e checar tais adaptações.

Palavras-chave: Auto-adaptação, auto-teste, loop de controle, ambiente dinâmico.

Table of Contents

1 Introduction

Today's complex software systems are able to autonomously work in dynamic environments by adjusting and adapting their behavior. The characteristics of the environments change unpredictably, forcing the systems to self-adapt and improve their own behavior while trying to cope with such changes.

In this context, self-adaptive systems become one of the main focal points of software engineers. Several approaches describing how systems can perform self-adaptation have been investigated. Such approaches propose self-adaptation processes or architectures able to check the need for the adaption and also to adapt the behavior when it is necessary. However, only few (Neto et al., 2009), (Stevens et al., 2007), (Weng et al., 2005) provide means to check if the adapted behavior is more adequate to the new characteristics of the environment than the previous one. In order to do so, it is necessary to test at runtime the adapted behavior by investigating its compliance with the new environment requirements.

The main problems of the approaches that test the adapted behavior at runtime are the following: (i) it is not possible to define different input data and output assertions to the tests, (ii) different log formats, to be useful when analyzing the results of the performed tests, cannot be defined, and (iii) the self-test activity that such approaches use is specifically defined for a given self-adaptation process, i.e., such activity cannot be used in any other process without great effort.

The goal of the paper is to present a self-adaptive process that contemplates an activity that promotes the test of the adapted behavior and can be used in any other process, i.e., a self-test activity that is process-independent. In order to do so, our approach extends the self-adaptive process proposed in (IBM, 2003). Such process is composed of four activities: (i) *monitor* that collects application data; (ii) *analyze* that analyzes those data by trying to detect problems, (iii) *plan* that decides what should be done in the case of problems; and (iv) *execute* that changes the application due to executed actions. This process (or control loop) was chosen since it clearly defines the activities related to the self-adaptation process and has been used as a basis by different approaches such as (Neto et al, 2009), (Stevens et al., 2007) and (Soeters and Van Westen, 1996) by making it easier to relate the works.

The self-adaptive process being proposed extends a basic control loop with the new activity that verifies (or tests) if the adapted behavior complies with the new requirements. In this paper we describe the dynamics of the extended control loop and detail the test activity.

The paper is organized as follows. In Section 2 we present related works. Section 3 introduces our proposed control loop while detailing its dynamics and the test activity. In Section 4 a case study is described to illustrate our approach. Section 5 concludes and describes some future work.

2 Related Work

(Denaro et al., 2007) present a self-adaptive approach for service-oriented applications by using a control loop structured in the following way: monitoring mechanisms, diagnosis mechanisms, and adaptation strategies. A prototype of the self-adaptive approach was implemented and consists of a pre-processing and a generation step. The pre-processing test is composed of the following phases: identification of possible integration problems, generation of test cases for revealing integration problems, and definition of suitable recovery actions. The main drawback of this approach is that the test activity being proposed (called pre-processing) does not consider important steps defined in our approach, such as the possibility to define input data and output assertions to be used in different actions. Besides, the self-test being proposed is fixed to the process, while our approach can be adapted to different control loops.

A framework for testing self-adaptive systems at runtime is proposed by (Stevens et al., 2007). The authors introduce the concept of an autonomic container. An autonomic container is a data structure that has self-managing capabilities, and also has the implicit ability to self-test. They have implemented a reduced version of this autonomic container and validated it using a prototype. The authors use a strategy that tests copies of managed resources while the actual managed resource is being used by an application, a technique known as *replication with validation*. However, such an experiment does not allow the use of the self-test activities in other architectures. The approach works together with IBM's layered architecture for autonomic computing (AC) systems (IBM, 2003), which consists of managed resources, touchpoints, touchpoint managers, orchestrating managers, and manual managers. Each test activity is connected with the layers and is implemented according to such layer characteristics. Therefore, it is not possible to use the proposed self-test activities in different self-adaptation processes.

(Wen et al, 2005) proposes a Software-Based Self-Test (SBST) framework in order to be used in microprocessors. The SBST proposal consists of two key steps: (i) self-test of on-chip processor core(s), and (ii) test and response analysis of other on-chip components, using the tested processor core(s) as the pattern generator and response analyzer. Since the components of the framework are tied in with concepts of microprocessors, it is not possible to use it in different domains, such as the approach being proposed in this paper.

3 Control Loop Applying Self-test

In this section we describe a new control loop of self-adaptation based on the model illustrated in Figure 1 (IBM, 2003). As stated before, this model has four main activities (*monitor*, *analyze*, *plan* and *execute*) that do not apply the self-test concept.

3.1 Self-adaptative + Self-test Control Loop

After adapting the behavior and before executing it, the agent may test if the self-adapted behavior fulfills the new environment requirements. In order to contemplate such task, a new control loop based on the self-adaptation process illustrated in Figure 1 and including a new activity called *test* was created. Figure 2 illustrates the interdependences among the five activities defined in the new control loop: *Monitor*, *Analyze*, *Decision*, *Test* and *Execute*, quickly introduced below. In the following sub-section the new *Test* activity is explained in more details.

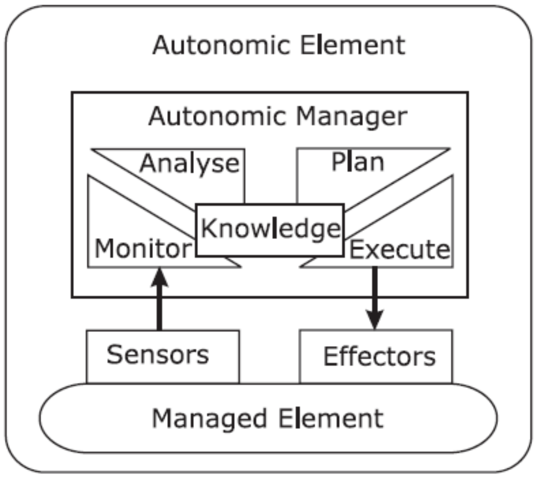


Figure 1. Model suggested by the IBM (IBM, 2003)

1. Monitor: The monitor activity receives through the agent sensor information coming from the environment. It is responsible for receiving, filtering and formatting the data provided by the sensor. After noticing the available data, the monitor filters and formats them to be manipulated by the other activities of the control loop. The filtered and formatted information is then made available to the analyze activity;

2. Analyze: The analyze activity is responsible for providing methods to analyze the data collected in the previous activity in order to detect problems (or necessities) and suggest new solutions;

3. Decision (called Plan in (IBM, 2003)): This is the third activity responsible for deciding which action (behavior, service, etc) will be the next one to be executed by the agent, while trying to achieve its goal. This decision is based on the information provided by the analyze activity. Depending on such information, it may be necessary to adapt the agent behavior. It is the aim of this activity to choose if the behavior should be adapted and to effectively adapt it by, for example, choosing an alternative action;

4. Test: Anytime an action is chosen, the decision activity can call the test activity to perform a set of tests to validate the selected action. The information about the successes or failures met by the tests is provided to the decision activity. The decision activity is then responsible to decide whether such action should be skipped or changed for another one. If a new action is chosen, the decision activity can call the test activity again to test such action;

5. Execute: This is the last step of the self-adaptation process. It receives the selected action from the decision activity, and informs the agent about the action to be executed.

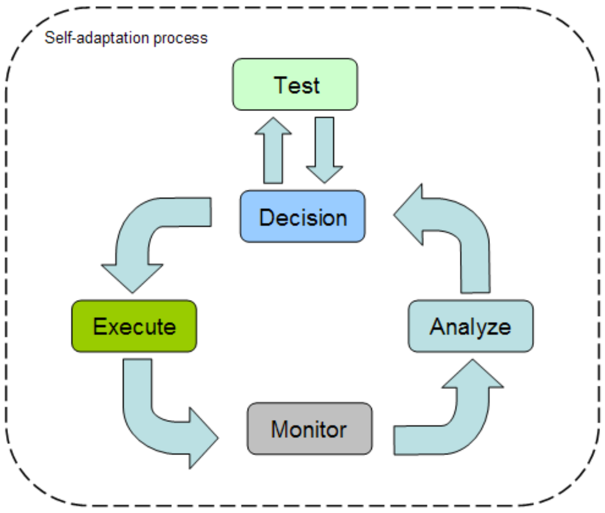


Figure 2. New control loop with test activity.

3.2 Test Activity

As mentioned previously, the test activity is responsible for testing the actions chosen by the decision activity and to inform the decision activity if errors have occurred when testing these actions. The test activity is composed of four steps. The first two steps should be executed off-line and the other two are executed together with the application:

1. In this step the application designer should relate the actions of the agent to the test cases used to test such actions. The set of test cases are predefined by the designer and related to the actions that they are able to test.
2. The next step defines the data to be used as input data and output assertions while testing the actions. A test case uses the corresponding input data to execute a given action and the associated output assertions to check if the results found after executing the task are the expected ones. Note that the same input data can be used in different actions, such as in cases that test different application versions.
3. After relating the test cases and the actions, and also defining the related data, the tests can be executed when requested by the decision activity. Therefore, the third step of the test activity executes the test *per si*. Different types of tests can be executed, such as unit test, functional test, performance test, etc.
4. In the sequence, it is time to generate the output logs with the results of the executed test. These logs will be used by the decision activity in order to decide to execute the action or choose another one.

In order to use the test activity in another control loop, it is necessary to:

1. Define the input and output assertions at design time;
2. Define at design time the type of logs that can be used to format the feedback provided by the test activity;
3. Associate the actions with the test cases that will be used to test them at design time;

4. Implement an activity able to analyze at runtime the logs provided by the test activity;
5. Call the test activity by providing the action that will be tested.

4 Case Study: Georisc

Landslides are natural phenomena, which are difficult to predict since they depend on many (unpredicted) factors and on the relationships between them. The annual number of landslides is in the thousands, and the infrastructural damage is in billions of dollars (Karam, 2005). Since there is a need to systematically deal with these factors, one of the main challenges faced by the specialists is to decide the most appropriate model configuration to generate susceptibility maps (SM), i.e., maps that show locations with landslide risks in a specific area. By using such a map, it is possible to identify the areas with highest risks in a region.

Considering this context, we implemented an application where a software agent has the goal to generate an SM that shows the places with landslide risks in Rio de Janeiro, a city in Brazil. The application agent tries to meet a susceptibility model that creates the best SM based on data provided by a user. The agent adapts its behavior while choosing the most appropriate model. Each model is implemented as a web service, and the agent uses OWL-S (Martin et al, 2009) files in order to find such models. OWL-S is an ontology, within the OWL-based framework of the Semantic Web, for describing Semantic Web Services.

4.1 Main Idea

The implemented system is composed of an agent called SM agent, as illustrated in Figure 3. This agent uses a (default) model provided by a given web service to create the SM. In the case a problem occurs while using the service or the input data that are needed to execute the service are not available, the agent can adapt its behavior by choosing another service. Several problems indicate that the agent should adapt its behavior, such as: the service cannot be accessed or the generated map is no good. In order to find different models available as web services, the application provides OWL-S files that can be used by the agent.

When a different service is chosen and before it is used, the agent tests if it is working satisfactorily – to be explained in the next section. The data to be used as input and output assertions while testing the web services are stores in a XML file. Figure 4 illustrates one of these files. In the example, the data are related to a test that verifies whether a susceptibility model generates the expected set of files. Such files contain information about landslides in a specific area.

Note that different web services can use different data. If the service is approved by the test, the agent behavior is adapted and such service is used to generate the SM.

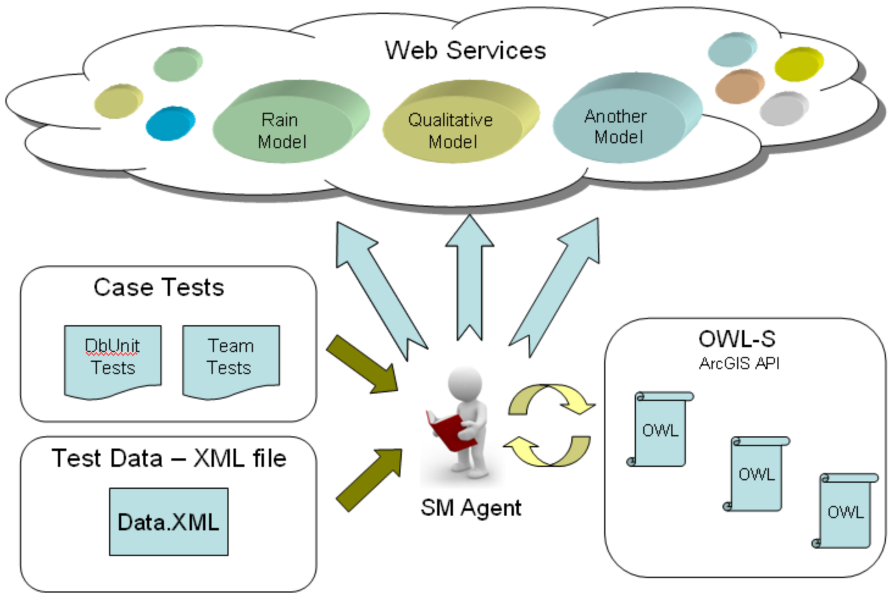


Figure 3. Georisc application.

```xml
<contexts>
    <context version="1.0" name_context="Landslides">
        <action name="test_files_created_Rain_Model">
            <inputs>
                <input name="TypeTool" value="AlgebraRain"/>
                <input name="setOutFilePath" value="C:/Dados/Shapefile/Rain/ActualDay/cda"/>
                <input name="InExpression"
                  value="[C:/Dados/Shapefile/Rain/ActualDay/cda] - Exp(-1.14 * Ln([C:/Dados/Shapefile/Rain/FourDay/cfd]) + 9.17)"/>
            </inputs>
            <outputs>
                <output name="file1" value="C:/Dados/Shapefile/Rain/ActualDay/cda/prj.adf"/>
                <output name="file2" value="C:/Dados/Shapefile/Rain/ActualDay/cda/dblbnd.adf"/>
                <output name="file3" value="C:/Dados/Shapefile/Rain/ActualDay/cda/hdr.adf"/>
                <output name="file4" value="C:/Dados/Shapefile/Rain/ActualDay/cda/log"/>
                <output name="file5" value="C:/Dados/Shapefile/Rain/ActualDay/cda/sta.adf"/>
                <output name="file6" value="C:/Dados/Shapefile/Rain/ActualDay/cda/w001001.adf"/>
                <output name="file7" value="C:/Dados/Shapefile/Rain/ActualDay/cda/w001001x.adf"/>
            </outputs>
        </action>
        ...
    </context>
    ...
</contexts>
```

Figure 4. XML file with input and output data to tests.

4.2 SM Agent

As stated previously, the SM agent is responsible for providing a satisfactory SM that is generated by *susceptibility models* implemented as web services. The parameters for a good SM are defined in a XML file. The SM is compared with an inventory map and these parameters are checked. The inventory map stores the history of landslides that have occurred in Rio de Janeiro over the last thirty years.

The default model executed by the agent is called the factor of safety (Soeters and Van Westen, 1996). Such model uses a large set of data, such as environment temperature, the accumulated rain in the area, and its vegetation type, to generate the SM. In the case the set of input data is not available, other models can be used, such as, the rain model (Soeters and Van Westen, 1996), which uses only the information about the accumulated rain in the area.

If a problem occurs while using the service or the input data that are necessary to execute the service are not available, the agent can choose to adapt its behavior by selecting another service. With the aim to adapt its behavior, the agent may use the provided control loop composed of the five activities described in Section 3: monitor, analyze, decision, test and execute.

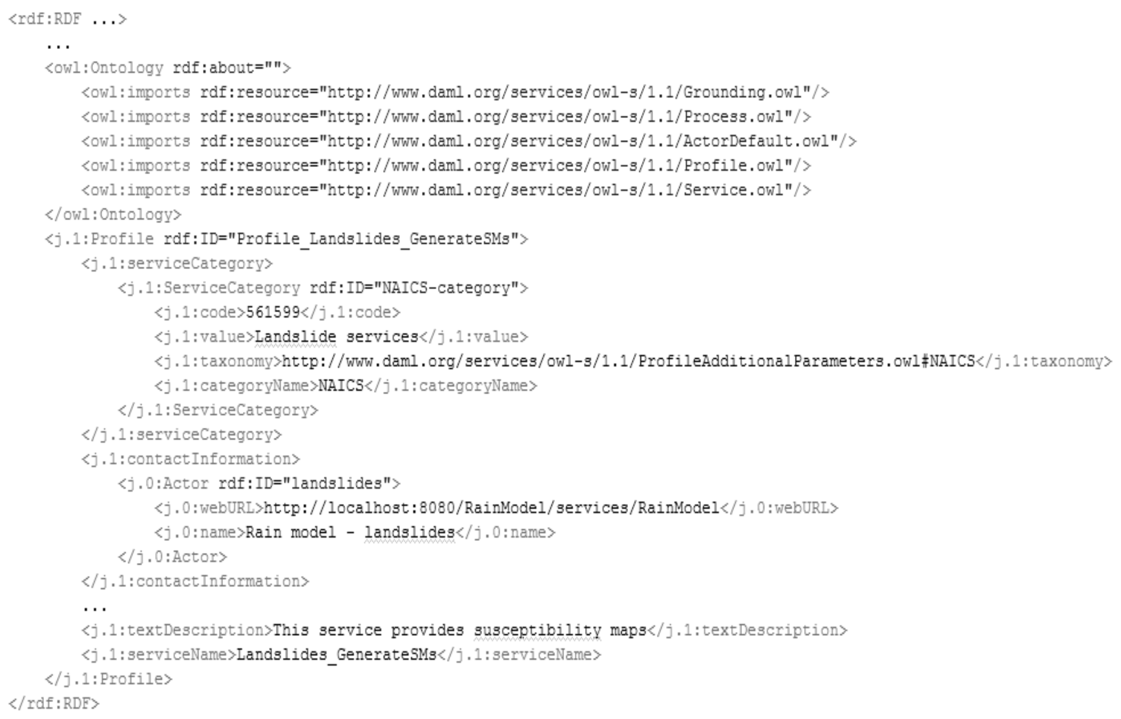
In the *monitor* activity the agent collects data provided by the user to be used by the *susceptibility model*. Next, in the *analyze* activity the agent selects the OWL-S files that can be used to find web-services able to create SMs. The OWL-S files describe the directions of the web-services, their names, the context where they execute, besides other information that can be used to identify them. Figure 5 presents part of an OWL-S file used in this example.

The *decision* activity receives the OWL-S files from the analyze activity and uses them to choose a web-service that implements the default model. The test activity initiates when the web-service is selected. Such activity executes a set of test cases and generates an output log with the result of the tests. Different test cases can be executed, such as:

1. One of the most simple test cases only verifies if the web-service is online;
2. Another very simple test case checks if the agent has the data required by the service to be executed. It can be checked by using the XML file where the input data used by the services are described;
3. A more complex test may compare the SM in the inventory map with the SM generated by the web-service to find out if the generated SM is satisfactory. Note that, since it is only a test, the web-service will provide only a sample of the map it is able to generate. The comparison is made based on this sample;
4. Another possible test checks how much time the service takes to generate the sample maps in order to estimate the time to generate the whole map. This may be important if the agent is seeking small response times.

Next, the *decision* activity is re-executed in order to analyze the log generated by the test activity. If the web-service is "approved" by the test cases, the decision activity adapts the agent behavior by choosing this web-service to be executed. If it is not the case, another web-service is selected from the OWL-S files and sent to the test activity to be tested.

In the last activity, called execute, the agent uses the chosen web-service that implements the susceptibility model that will generate the SM.



```
<rdf:RDF ...>
    ...
    <owl:Ontology rdf:about="">
        <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Grounding.owl"/>
        <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl"/>
        <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/ActorDefault.owl"/>
        <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Profile.owl"/>
        <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Service.owl"/>
    </owl:Ontology>
    <j.1:Profile rdf:ID="Profile_Landslides_GenerateSMs">
        <j.1:serviceCategory>
            <j.1:ServiceCategory rdf:ID="NAICS-category">
                <j.1:code>561599</j.1:code>
                <j.1:value>Landslide services</j.1:value>
                <j.1:taxonomy>http://www.daml.org/services/owl-s/1.1/ProfileAdditionalParameters.owl#NAICS</j.1:taxonomy>
                <j.1:categoryName>NAICS</j.1:categoryName>
            </j.1:ServiceCategory>
        </j.1:serviceCategory>
        <j.1:contactInformation>
            <j.0:Actor rdf:ID="landslides">
                <j.0:webURL>http://localhost:8080/RainModel/services/RainModel</j.0:webURL>
                <j.0:name>Rain model - landslides</j.0:name>
            </j.0:Actor>
        </j.1:contactInformation>
        ...
        <j.1:textDescription>This service provides susceptibility maps</j.1:textDescription>
        <j.1:serviceName>Landslides_GenerateSMs</j.1:serviceName>
    </j.1:Profile>
</rdf:RDF>
```

Figure 5. OWL-S file example.

5 Conclusions and Future Work

This paper presents a self-adaptive process that takes into account a self-test activity. Before adapting the agent behavior it is important to check if such behavior contemplates the environment requirements.

In order to demonstrate the use of our proposed process, we have used it to assist in the adaptation of agents that make use of web-services. The agents adapt their behavior while selecting different web-services. Before such adaptation, the service is tested. The agent behavior will only be adapted, i.e., the service will only be used, if the test cases have approved it.

We are in the process of developing an application framework that implements the self-adaptation +self-test process by making available several extension points to be defined by the application itself. By using such a framework, the application agents will be able to use test cases predefined by the framework and also others defined at design time by the application designer.

References

Denaro, G., Pezze, M., and Tosi, D., 2007, Designing Self-Adaptive Service-Oriented Applications. In Proceedings of the Fourth International Conference on Autonomic Computing. International Conference on Autonomic Computing. IEEE Computer Society, Washington, DC, 16.

Fayad, M. , Johnson, R., Building Application Frameworks: Object-Oriented Foundations of Framework Design (Hardcover), 1999,Wiley publisher, first edition, ISBN-10: 0471248754, 1999.

Georisc, Last access at July 2009, http://wiki.les.inf.puc-rio.br/index.php/Georisc.

IBM. 2003, An architectural blueprint for autonomic computing. Technical Report., IBM.

Karam, K. S., Landslide Hazards Assessment and Uncertainties, 2005, Thesis: Massachusetts Institute of Technology.

Kephart, J. O. and Chess, D. M., 2003, The Vision of Autonomic Computing. *Computer* 36, 1 (Jan. 2003), 41-50.

King, T. M., Babich, D., Alava, J., Clarke, P. J., and Stevens, R., 2007, Towards Self-Testing in Autonomic Computing Systems. In Proceedings of the Eighth international Symposium on Autonomous Decentralized Systems. ISADS. IEEE Computer Society, Washington, DC, 51-58.

King, T. M., Ramirez, A., Clarke, P. J., and Quinones-Morales, B., 2008, A reusable object-oriented design to support self-testable autonomic software. In Proceedings of the 2008 ACM Symposium on Applied Computing (SAC). Fortaleza, Brazil ACM, New York, NY, 1664-1669.

Martin, D., et. al. ,OWL-S: Semantic Markup for Web Services, Last access at July 2009, http://www.w3.org/Submission/OWL-S/.

Mengusoglu, E., Pickering, B., 2007, Automated management and service provisioning model for distributed devices, Proceeding of the 2007 workshop on Automating service quality: Held at the International Conference on Automated Software Engineering (ASE), New York, USA, pp38-41.

Neto, B. F. S., Costa, A. D., Netto, M. T. A., Silvia, V., Lucena, C. J. P., July 2009, JAAF: A Framework to Implement Self-adaptive Agents. In Proceeding of the 21st International Conference on Software Engineering Knowledge Engineering (SEKE'2009), Boston, Massachusetts, USA, pp. 212-217.

Soeters, R. and Van Westen, C.J., 1996, Slope Instability Recognition, Analysis and Zonation. In: Turner, A.K. and Schuster, R.L. (eds). Landslides, investigation and mitigation. Transportation Research Board, National Research Council, Special Report 247, National Academy Press, Washington D.C., U.S.A., p 129-177.

Stevens, R., Parsons, B., and King, T. M., 2007, A self-testing autonomic container. In Proceedings of the 45th Annual Southeast Regional Conference (Winston-Salem, North Carolina). ACM-SE 45. ACM, New York, NY, 1-6.

Dobson, S., Denazis, S., Fernández, A., Gaiti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F. 2006, A survey of autonomic communications, *ACM Transactions Autonomous Adaptive Systems (TAAS)*, 1(2):223{259, December.

King, T. M., Ramirez, A. E., Cruz, R., Clarke, P. J. 2007, An integrated self-testing framework for autonomic computing systems, Journal of Computers, Vol. 2, No. 9, November.

Web Services Activity, Last access at July 2009, http://www.w3.org/2002/ws/.

Wen, C., Wang, L.-C. Cheng, K.-T, Yang, K. Liu, W.-T.., May 2005, "On A Software-Based Self-Test Methodology and Its Application". IEEE VLSI Test Symposium.