J. Comp. Int. Sci. (2019) 10(3):-97-110
http://epacis.net/jcis/PDF_JCIS/JCIS-0164.pdf
jcis@epacis.net

PAN-AMERICAN ASSOCIACION OF COMPUTATIONAL INTERDISCIPLINARY SCIENCES

A Framework To Provide Self-Adaptive Support To Scientific Software Systems

Moacyr Goncalves Cereja Junior [a], Nilson Sant'Anna [a] and
Marcos Paulo Salgueiro Moraes [a]
[a]National Institute for Space Research, Sao Jose dos Campos, SP, Brazil

Received on Dez, 2018 / Accepted on Fev , 2019

Abstract

Currently, INPE space weather applications process data from different instruments. To keep the systems operating in order to deliver the expected results, challenging issues arise: How to measure the instrument's quality of service? How to adapt the system to continue operating in the event of unavailability? Self-adaptive systems are presented as a solution to adapt the system in response of changes in their operating environment. In this context, a framework is presented to provide self-adaptive support to software systems. The framework called "Arctic Fox" supports the entire adaptation life cycle: Monitoring, Analysis, Planning and Execution. An adaptation is triggered after metrics analysis and detection of service level agreements (SLA) violations, if so, an actuator (Executor) is activated in the system to promote the necessary architectural adaptations to keep the system serving the purpose for which it was designed.

Keywords: Space Ground Systems, Self-Adaptive Software, Service Level Agreement, Quality of Service, Resource Optimization.

1. Introduction

As information technology advances in the world, it is also notorious the increase of demand for sophisticated and complex scientific applications. These applications also requires robustness, effectiveness and efficiency in the acquisition process, persistence and information distribution . The concept of Big Data is not utopia anymore , there is already a growing reality in professional organizations that works with scientific data.

INPE (National Institute for Space Research) has in its Institutional Goals, a Program for Space Weather - Embrace. This Program intends to be Regional Operational Center for Space Weather. Through the services provided by this center, Scientists, Researchers, Engineers, companies and organizations in general can use, view and cross-match information produced by several scientific instruments that produce satellite images in

its various bands, ionosphere profiles produced by satellites, GPS networks data, ionosondes data, magnetometer data, radio data, and X-ray and solar flares data, among others.

A Space Program, in simplified form, has Space and Terrestrial structures [1]. The space structures referred to as Space Segment consists of satellites, components, instruments that produce data (payload) and communication systems with the earth. The Ground Segment (terrestrial structures) consists of earth stations, operation center and Satellite Control (Spacecrafts) Center, Payload Operation Control Center and Mission Control Center.

In this paper we will focus on the Payload Operation Center which among other things has a Management and Dissemination Information System - MDIS. Develop systems like this with professional and technical characteristics is not a trivial task and requires a multidisciplinary team of Systems Engineers, Software Engineers, Software Architects, Application Architects, researchers in Technologies and several space scientists.

Once in operation, it is necessary to ensure high effectiveness, efficiency and reliability of services, this is achieved through two strategies: service redundancy and / or replication. Here we note that redundancy and replication does not refer to equipment and computers, but in services (software) that run on a set of machines already optimized through scaling and virtualization of computers that perform load balancing. Another important aspect to ensure these performance characteristics, efficiency and reliability is to use the concept of Quality of Service. By establishing quality parameters, contracts between provider / consumer and constant monitoring of the quality of services is possible to know, at all times, the degree of reliability that the services are operating and thus take strategic actions to ensure the desired effectiveness, reliability and efficiency for a MIS

In this paper we propose the use of Arctic Fox Framework by INPE Space Weather Program. The Arctic Fox is a framework of computational resources capable of:

- Orchestrate optimally (self-adaptation), ie, manage, organize the services provided by Space Weather and create compositions.,
- To enable the establishment of quality attributes and agreements among producers/consumers
- real time monitoring of Space Weather services

  The paper is organized as follows: first part presents the current state of the Space Weather Management Information System and the rea-

sons which leads to its development. In a second part, the requirements and the architectural aspects of the ArticFox Framework and the proposal for the the HiPeR EMBRACE system (High Performance and Reliable Embrace System)

2. Space Weather Architecture

This section present the fundamental premises and problems related to EMBRACE architecture [2] [3] [4]. Premise 1: There are too many instruments and sensors producing different data at different rates and volumes. As an example, the GPS reception stations network as shown in Figure 1.

Figure 1 - GPS receiver stations at EMBRACE and TEC Map application (Total Electron Content. [Source: Space Weather].

Problem 1: If INPE uses standard (FTP) file transfer protocol to receive data, multiple copies of the same data file is generated on Space Weather servers in Sao Jose dos Campos. These copies distributed among multiple servers generate delays, storage inefficiency and data interference.

Solution 1: To resolve this issue, a PIPELINE solution was implemented to deliver data directly to Space Wheater servers. These solution is responsible for persisting data and was deployed at EMBRACE to manage instruments data reception.

Other issues include:

- Different models and applications require specific data preparation to be effective and efficient.
- Once the data is persisted, is necessary to processes the raw data to produce derived data by asynchronous processing, this is made by a data and meta-data subsystem which manipulate, process and filter data.

- Provide space weather data to other applications and end users through web services.
- Information exchange standards as virtual observatories, can be taken into account in architecture design.
- There are a lot of data arriving at the data repository (acquisition) and a large number of users and applications using the data (spread). To this we must design a good data persistence model (Logic Model) and deploy a set of servers (computers) installed in a scalable way, ie increases and decreases depending on demand (Physical Structure of Computers).

Figure 2 shows a simplified architecture of the current space weather.

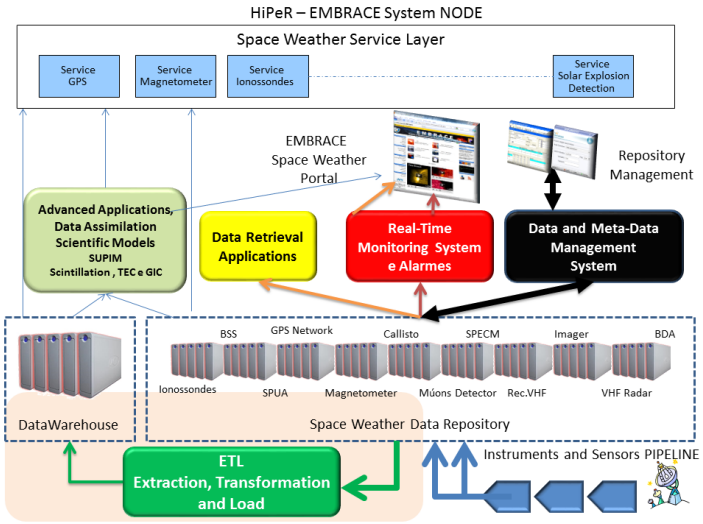

Figure 2 - Space Weather Architecture.

3. Management Information System Requirements For High Efficiency and Reliability

High reliability and high efficiency are scientific data management system requirements which must be satisfied. The EMBRACE architecture is based on redundancy and replication of components implemented as Web Services. Redundancy means that the system may have replicated services or other

ones with similar features distributed among several servers and locations. These components may have different quality attributes like response time, availability, up-time, if so, which one is better to satisfy user quality of service requirements?

An orchestrator is a computational agent that organizes composition of these web services. Web Service calls can be changed at run time based on certain execution strategies based on certain quality attributes. Exchange of components are called computational adaptation.

Figure 3 shows a representation of a Primary Service component and the components execution order (alternative services can replace original service).

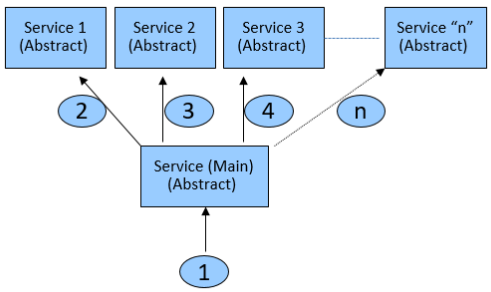

Figure 3 - Service composition with numbered callout order.

These services may be able to adapt through redundancy and / or replication. One service may have two (2) or three (3) implementations, so any of them can be used at run-time. Soon the Orchestrator should at the right time (when), choose the service that will be executed (Who) based on some criteria or heuristic (as). Contextual Knowledge is key for the orchestrator dynamically optimize a new composition.

Then: "Self-Adaptive Software modifies its behavior in response to changes in its operating environment by operating environment, we understand any element observable by the software system, ie, user input, external hardware devices and sensors and instrumentation programs.". [5]

"Adaptation: Modification of a software system to meet new needs and changing circumstances". [6].

"Adaptability is the ability of a software system to meet new requirements and adjust to new operating conditions during its operating time".[7]

In the view of [8] self-adaptive software is a closed loop system retro-fed by its context and itself. In this article the concept of self properties of software systems to obtain the self-adaptivity of the system is presented. The properties are self-organized in a hierarchy structure.

The self-properties is organized on three levels on hierarchy. Self-adaptivity and self-organization are general properties which are decomposed into primary and primitive properties on two different levels. Below is a description of each layer and property:

- **General Level**: This level contains global properties of self-adaptive software. The subset of these properties below the self-adaptability hierarchy consists of self-management, self-government, self-maintenance, self-control and self-assessment. Another subset of this level is self-organization.
- **Main Level**: The autonomic computing initiative, IBM defines a set of four properties for this level raised to paritr biological self-adjustment mechanisms. For example, the human body has similar properties, so as to adapt to changes in their environment such as temperature change in the environment or a lesion or of a failure of internal organs. The following list elaborates a bit more about the details of the properties:
  - **self-configuring**: is the automatic and dynamic configuration capabilities in response to changes, that is, install, update, integrate and compose / decompose software entities;
  - **self-healing**: is linked to self-diagnosis or self-repair is the ability to discover, diagnose and react to disturbances. It can also anticipate potential likelihood of problems and thus take appropriate measures to prevent a failure. Self-diagnosis refers to diagnosis of errors, defects and failures, while self-repairing relates to recover from them;
  - **self-optimizing**: is also called self-adjusting or self-regulation is the ability to manage the resource allocation and performance to meet the needs of different users. Response time end to end flow, use and load are examples of important issues related to this property. IBM's definition indicates auto optimization as "ability of IT environment efficiently maximize the allocation and use of resources to meet user needs with minimal intervention Self-optimization addresses the complexity of system performance management.";

  - **Self-stabilization**: it is the ability to detect security breaches and recover from its effects. It has two aspects, that is, defend the system against malicious attacks, and anticipate problems and take steps to avoid them or to mitigate their effects.

- **primitive level**: Self-knowledge (self-awareness), self-monitored, self-located, and context awareness (context-awareness) are primitive properties discussed below in more detail:

- **Self-awareness**: means that the system is aware of their states and behavior. This property is based on self-monitoring property that reflects what is monitored.

- **Context-Awareness**: means that the system is aware of its context, which is the operating environment.

In addition to the self-properties, more five self-properties can be seen at [9] dissecting self-properties paper, they are: Self-stabilization, Self-organization, Self-Scaling, Self-immunity, Self-containment.

3.1. Approaches to Software Adaptation

In figure 4 is presented a methodology for adaptation that was suggested in [8]. The upper half of the diagram, named adaptive management (management adaptation) describes the life cycle of adaptive software systems. The life cycle can have people in the loop or be completely autonomous. Descriptions of each life cycle stage of the loop management adaptation can be seen below:

- **"Enact changes and collect observations"**: collecting observations, i.e., metrics of the environment and passes this information to the next phase, also activates or approve the changes envisaged in Deploy including at least phase, performance monitoring, security inspections and check constraints. descriptions changes;

- **"Evaluate and monitor observations"**: it refers to all forms assess and monitor the execution of an application;

- **"Plan Changes"**: refers the task to accept the assessments, define an appropriate adaptation and build an implementation plan for the implementation of adaptation.

- **"Deploy change descriptions"**: is the coordinated conduct of descriptions of changes, components and possibly new observers or evaluators for the effective implementation of adaptation in the system.

On the other hand the implementation of changes can also extract data and possibly application components running and lead them to some other point analysis and optimization.

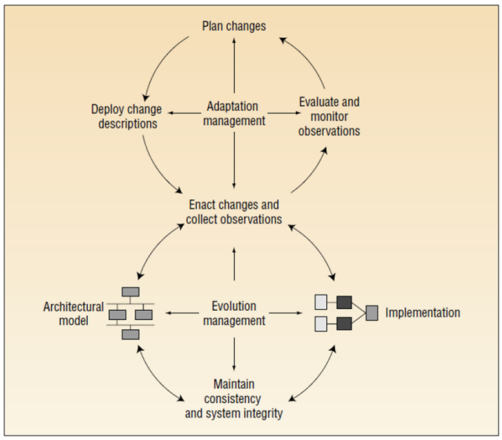

Figure 4 - High-level processes for a comprehensive and general purpose for adaptive software systems. [Source: [8]]

Systems adaptations must be autonomous, i.e., without human intervention. To achieve this, the system should be monitored, analyzed, and in the case of abnormality detection, a plan for adaptation should be prepared and executed. So, the adaptation life cycle is based on monitoring, analysis, planning and executing adaptation.

4. Arctic Fox and Arctic Fox QoS

Frameworks [8] model a specific domain or an important aspect of it. They represent the domain as an abstract design consisting of abstract classes (or interfaces). The abstract design is more than a set of classes, because it defines instances of classes and how then collaborate with one another at runtime. A framework contains reusable implementations in the form of abstract and concrete classes [3].

In order to provide a mechanism to adapt software systems is proposed a framework for adaptation with capacity of monitoring, analysis, planning and implementation of changes in software systems. The proposed framework shown in Figure 5 was named "Arctic Fox", an animal with interesting adaptation properties, it adapts its coat to survive temperatures so low that can reach -58 F (-50 C).

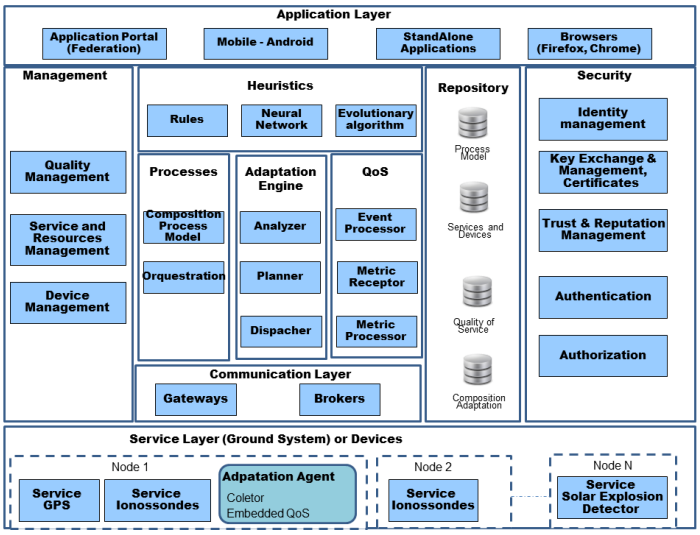

Figure 5 - Arctic Fox Framework.

The Framework has a set of components distributed in 3 layers. A description of main elements are given below:

- **Software System**: The system that can be adapted and kept under supervision.
- **Arctic Fox Adaptation Agent**: installed on ground system. This component board an API with facilities to develop Service Level Agreements (SLAs Impl); define and collect metrics (Collector); discover and register agreements (Discovery) and deploy adaptation plans at Software System (Adaptation Deployer).
- **Arctic Fox Framework Broker**: Layer responsible for receive and calculate metrics, analyse data, plan adaptation and dispatch the plan to be deployed into software system. The main components are:
  - **Metric Receiver**: receive metrics collected by the agents;
  - **Metric Processor**: calculate derived metrics;
  - **Event Processor**: start analysis process;
  - **Analyzer**: try to detect SLA violations, therefore, this component is composed by abstract classes and interfaces, the analysis task is delegated to a concrete implementation located at Heurist Layer.

– **Planner**: Plan adaptation tasks to be executed on Ground System. It receives a pre-adaptation plan from analyzer, define tactics, order and an agent to execute the plan, after that, it publishes the plan for deploy.
– **Dispatcher**: Send adaptation plan to Adaptation Agent.
– **Heuristic Layer**: has analysis engines concrete implementations.
– **Rule Based Engine**: Implementation of a rule based adaptation engine composed by a rule base and a fact base necessary to analyse events triggered by the framework.

```
package br.inpe.arcticfox.api.registry;

import br.inpe.arcticfox.api.annotations.*;

@SLAContract(
    name = "SLA Contract for Service A",
    description = "SLA Contract for Service A",
    priority = SLAPriority.HIGHLY_IMPORTANT,
    status = SLAStatus.ACTIVE,
    architectureElement =
    @ArchitectureElement(
          name = "ServiceA",
          type = ArchitectureElementType.COMPONENT,
          properties = { @Property(name = "endpoint", value="http://service.com/ServiceA"), @Property(name =
"protocol", value="SOAP") }
    ) )
public class ServiceAContract {
          @Tatic ( matchPolicy = MatchPolicy.MATCH_ALL_CONDITIONS,
          consequenceOf = {
          @SLAViolationCondition(metric="THROUGHPUT", operator = MetricOperator.GREATER_THAN, value = 50) })
          public void switchToServiceB() {
          //Tatic is an architectural operator...
          }
}
```

Figure 6 - Service Level Agreement java implementation.

6. Case Study

To validate the concepts used in the framework, a small prototype with two Web Services simulating a high load scenario was developed. The PoC try to detect violation of SLA throughput metric and redirect to an alternate service. Description of the proof of concept test scenario: 1. Java SLA contract implementation of an SLA throughput for the service A, 50 TPS (transactions per second) condition.; 2. In the event of SLA violation, requests should be redirect to the service B; 3. Stress test on the service A; 4. Real-time measurement of service workload; 5. Detection of Violation and Adaptation on-the-fly to service B.

A Java class implementation fragment example of SLA contract, developed with the framework API, can be seen in Figure 6.

```
18:08:45,293 INFO  [stdout] (http--127.0.0.1-8080-7) REQUESTS COUNT  -> 18378
18:08:45,294 INFO  [stdout] (http--127.0.0.1-8080-7) MEASURED THROUGHPUT -> 304.0
18:08:45,296 INFO  [stdout] (http--127.0.0.1-8080-1) MEASURED THROUGHPUT -> 304.0
18:08:45,297 INFO  [stdout] (http--127.0.0.1-8080-1) /articfox-webapp-poc
18:08:45,298 INFO  [stdout] (http--127.0.0.1-8080-10) REQUESTS COUNT  -> 18379
18:08:45,299 INFO  [stdout] (http--127.0.0.1-8080-7) /articfox-webapp-poc
18:08:45,308 INFO  [stdout] (http--127.0.0.1-8080-10) MEASURED THROUGHPUT -> 304.0
18:08:45,309 INFO  [stdout] (http--127.0.0.1-8080-10) /articfox-webapp-poc
18:08:45,309 INFO  [stdout] (http--127.0.0.1-8080-1) ChechServiceAThroughputViolation: SLA Violation
Found for ServiceA
```

Figure 7 - Application logs.

7. A Proposal for Space Weather - hyper-Embrace (high performance and Reliable Systems Embrace

After eight years of Space Weather System initial project you should advance to meet the complementary and non-functional requirements such as functional efficiency and reliability. As complementary functional requirements can be cited management services and how to use a standardized way services. As non-functional requirements we can mention the increased reliability by creating a replication and / or redundancy in other locations as INPE in Cachoeira Paulista, Santa Maria and Natal. As previously mentioned, the idea is to create a self-adaptive system that responds to external calls to INPE and thus assemble a composition of services using the best option based on quality attributes and determined adaptation criteria.

By using an abstract service provision of magnetic field data in the last 24 hours (magnetic disturbance index South American KSA) the system can use one of the implemented concrete services (any of the INPE nodes) without the application / user needs to know what INPE service is actually being used. Thus Figure 8 shows the hyper - Embrace System, a High Performance and reliability system for the EMBRACE system. The Arctic Fox framework has a key role in this architecture, its responsible to adapt an orchestrate service compositions deployed at EMBRACE cloud.

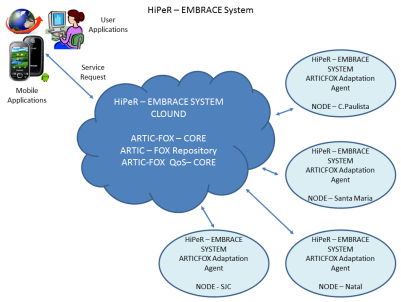

Figure 8 - Hiper EMBRACE System - High Efficiency System and Reliability of Space Weather.

8. Conclusions and Future Work

The paper presented the current scenario of INPE EMBRACE program for space weather applications, the organization of the current architecture EMBRACE architecture and an approach to the evolution of this in order to obtain more reliability and robustness. To implement these requirements were presented the concepts related to dynamic software adaptation and a framework that provides these features to a software systems.The framework presented has several components and can be used to optimize the Space Weather application services performance.

An architecture with its components and responsibilities for a dynamic adaptation framework was presented. It was also presented a proof of concept to validate the framework technology and feasibility. The framework continues in development at an advanced stage.

The paper also presented a proposal for a new Space Weather Architecture which incorporates the Arctic Fox Framework to satisfies reliability and adaptability requirements.

The next steps are to complete the development of framework main components and simulate a EMBRACE Space Weather real case.

References

[1] SantAnna N. Takahashi H.,Denardini C., Ivo A., Gomes V., Pereira F. Moraes M. Deployment Architecture to support the IT Capabilities of Brazilian Space Weather Program, in Anais Tenth European Space Weather Week, Belgium, 2013

[2] SantAnna N., Guerra E., Ivo A.,Pereira F.,Moraes M. Gomes V., Veras L.G. Modelo Arquitetural para a Coleta, Processamento e Visualizao de Informaes de Clima Espacial. Anais do X Simpsio Brasileiro de Sistemas de Informao,pg.125-136- Londrina Brasil, 2014

[3] Silva, J.D.S.D, SantAnna N., Takahashi H. Arquitetura do Sistema SISCLIMA para Armazenamento, disseminao de Dados do Clima Espacial do INPE. In Anais III Simpsio Brasileiro de Geofsica Espacial e Aeronomia, 2010.

[4] Oreizy, P. ; Gorlick, M.M. ; Taylor, R.N. ; Heimhigner, D. ; Johnson, G. ; Medvidovic, N. ; Quilici, A. ; Rosenblum, D.S. ; Wolf, A.L.;

An Architecture-Based Approach to Self-Adaptive Software, Intelligent Systems and their Applications, IEEE, Volume:14, Issue: 3, DOI: 10.1109/5254.769885, 1999 , Page(s): 54 62

[5] Taylor, Richard N. Software Architecture: Foundations, Theory, and Practice. John Wiley and Sons, 2008.

[6] Salehie M.; Ladan Tahvildari L. Self-adaptive software: Landscape and research challenges, ACM Transactions on Autonomous and Adaptive Systems (TAAS), Volume 4, Issue 2, Pages 14, ACM, 2009.

[7] Berns, A.; Ghosh, S.; "Dissecting Self-* Properties", Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems, IEEE Computer Society, San Francisco, CA, USA, 2009

[8] Riehle, D.; "Framework Design: A Role Modeling Approach". Ph.D. Thesis, No. 13509. Zrich, Switzerland, ETH Zrich, 2000. Disponvel online em http://dirkriehle.com/computer-science/research/dissertation/index.html, (acessed 24/05/2015).

[9] Blanchet, W.; Stroulia, E.; Elio, R. "Supporting adaptive web-service orchestration with an agent conversation framework. In Proceedings of the 2005 IEEE International Conference on Web Services, 541-549. July 11-15, Orlando, USA, 2005