Component-based Framework for Real-Time Systems using Design Pattern Approach for Interoperability

Emilia de Menezes Colonese[1]; Rovedy Aparecida Busquim e Silva[2]; Adilson Marques da Cunha[3]
Brazilian Aeronautics Institute of Technology – ITA Praça Marechal Eduardo Gomes, 50 São José dos Campos – SP – Brazil 12228-900

Abstract - **Component-based software engineering offers a way to solve complex systems by dividing them into the well-defined modules. Self-adaptive mechanisms are crucial to enable run-time reconfiguration and increase system components reuse in other systems. These systems must satisfy functional and non-functional requirements. Despite efficient data integration being a common aspiration, to achieve interoperability remains a challenge to implement the system's functional and non-functional requirements. For other components to work together with existing ones, and for the development of new system components to operate seamlessly with and among other systems, the adoption of a common set of "building codes" is required. This paper proposes a framework for real-time systems with data interoperability through a scope analysis of stakeholders' requirements. It implements the generic behavioral models for system Servers and Invokers. Changes of state diagram dimensions through integration or specialization, adapt the Invokers to the Interoperability Pattern, and the target system to the framework, leading software engineers to a transparent development and integration process. The framework can lead software components to high degrees of cost-effective reuse. This approach is tested in a real-time system prototype developed in the Brazilian Aeronautics Institute of Technology. The framework focused on dynamically activation of service components at run-time, self-adapting to external events. At the end, functional requirements and the software architectural structure are enforced such that the end-to-end timing behavior of the resulting system and its specifications can be verified.**

Key-Words – **Real-Time Systems, UAV, MDD, UML, Design Pattern, Interoperability, Self-Adaptive.**

I. INTRODUCTION

One major benefit of the object-oriented paradigm is the inherent support for abstraction centric, reusable, and adaptable design. In particular, it is common to construct complex systems using pre-defined frameworks. A framework is a collection of collaborating classes that provides a set of services for a given domain [1]. A developer customizes the framework to a particular application by subclassing and composing instances of the framework classes [11]. According to Booch [1], frameworks represent object-oriented reuse. The most important advantages of using frameworks can be listed as follows:

- The target system need not be written from scratch since it reuses the elements of the framework
- Frameworks structure the design of the target system by providing a set of predefined abstractions given by the classes in the framework. These classes provide an architectural guidance for the system design.
- Frameworks are open designs because their classes may be customized via subclassing.

1 colonese@ita.br; 2 rovedy@iae.cta.br; 3 cunha@ita.br

This work shows the conceptualization, implementation, and deployment of a framework for real-rime systems allowing interoperability among components, subsystem or systems, and activation of services at run-time.

The framework modeling follows a top-down approach and has self-adaptive characteristics that enable a activation of pre-defined services based on received external events, which are managed by use cases.

Adaptation in itself is nothing new, but it generally represents an ad hoc activity, involving future execution of condition forecasting at design time embedding adaptation decisions in the system code [7, 12, 14].

Self-adaptive characteristics allow user benefits, limiting run time overhead, reducing developer burdens, supporting instantiation of the framework module and, exploiting reuse and separated concerns [6].

The proposed approach is built on the following insights:

- Monitoring the correct deployment of system requirements;
- Offering an effective interoperability capability that is easily applied to others components, subsystems or systems; and
- Providing self-adaptive services at run-time according to received external events.

The case study shows that instead of developing proprietary solution, a real time system is easily integrated into the proposed framework.

This paper is organized as follows. Section 2 describes the main approach. Section 3 presents infrastructure used to model and develop the proposed framework. Section 4 presents the interoperability framework, and Section 5 describes the case study applying a target real-time system into the framework validating the solution. Finally, Section 6 concludes the work and summarizes the major findings.

II. APPROACH

The framework architecture is structured in three modular views. They represent the entire system based on different concern areas, which are defined by stakeholders' viewpoints: Logical (functionalities), Interoperability (reusability), Technical (infrastructure / feasibility) [7]. Figure 1 shows the proposed architectural structure.

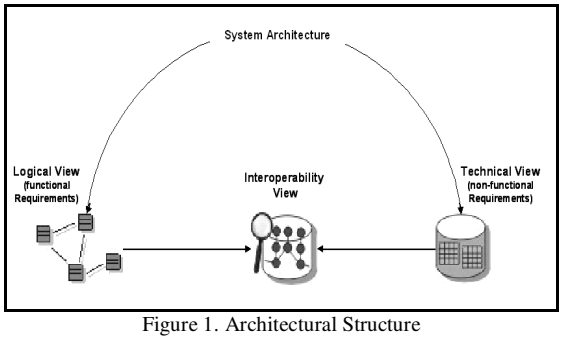

Figure 1. Architectural Structure

According to the four phases of the Unified Process [10], during the inception phase the system's requirements are identified. In the elaboration phase those requirements are mapped to business activities producing a "wish list" of systems functionality and infrastructure, named functional and non-functional requirements.

The system requirements might be presented as use cases. A use case may contain a coherent and cohesive set of detailed requirements. It describes how users, often referred to as actors, will interact with an application and what information they need from the system to accomplish their task. Full analysis of a use case will lead to a business processes definition (tasks) to accomplish the requirement(s), the interface with the actors, and data needed to perform tasks. Use cases diagrams are important to visualize the overall system architecture, specifying, organizing and modeling the system's behaviors.

Therefore, the Logical View covers a business representation of the system, which is acquired through functional requirements specification.

Framework is a "partially completed application" that customizes a specific application, and it is implemented by the Logical View.

The Technical View is concerned with non-functional requirements or Quality of Service (QoS), used to achieve as well as possible the functional aspects [3].

A use case may combine functional and QoS requirements. The authors introduced the Interoperability View to capture specific reusability of a transparent and common communication protocol among systems, subsystems or components, avoiding the use of a Data Translation Process [4]. The Interoperability View takes advantage of well defined system use cases. The choice of activating services depending on the use case that the invoker's event needs. The proposed framework focuses on the logical and Interoperability Architectural views.

A Design Pattern is a generalized solution to a commonly problem occurring. The authors observed a common pattern used for interoperability among components and subsystems during the design phase. The process of discovering a specific pattern is called Pattern Mining [10].

Therefore, the Interoperability View is a design pattern that implements specific property that is not delivered by the analysis model, named Interoperability Pattern. The design model differs from the analysis model in the way that it contains aspects that are not required but are included to make the entire system works better. In this approach the pattern is used to solve the interoperability issue among systems, subsystems and/or components.

III. FRAMEWORK'S NOTATION, MODELING, AND TOOL

A. Unified Modeling Language for Real Time

Modeling is an essential part in any software development, which allows communication links between system analysts and stakeholders in a high level of abstraction. The model effectiveness is translated into a low cost and reduced timeframe of software development.

During last years, Unified Modeling Language (UML) has been playing an important role for modeling Object-Oriented (OO) languages. The Object Management Group (OMG) characterizes UML as "a general-purpose modeling language for specifying, visualizing, constructing and documenting artifacts of software systems, as well as for business modeling and other non-software systems" [16].

Unified Modeling Language for Real Time (UML-RT) [19] was introduced to allow real-time systems modeling. UML-RT defines a constructor set based on the Real-Time Object-Oriented Modeling (ROOM) [20], which included capsules, connectors and ports to complement traditional UML behavioral components.

B. Model Driven Architecture

Model Driven Architecture (MDA) is an approach for using models in software development. This approach allows the translation of user's requirements to use case diagrams, and the system behavior to be separated from the implementation details [15, 16]. MDA enables the application to be easily ported from one environment to another by first creating one or more Platform Independent Models (PIM) and then translating the PIM into one or more Platform Specific Models (PSM).

By using an independent modeling language, such as UML-RT, MDA can reduce the development cycle of a real-time system while reaching platform independency. The MDA methodology provides software components portability, communication, and reuse, by using architectural process based on context partitioning.

C. Computer-Aided Software Engineering Tool

This work uses the Computer-Aided Software Engineering (CASE) tool from Rational, called IBM Rational Rose RealTime (RRRT). It is a visual real-time modeling tool, which includes the constructor set capable of supporting processes, methodology, common patterns and frameworks. It also generates code to test the designed system model [10]. RRRT components includes packages, passive classes (standard object-oriented classes), capsules (active classes with ports and connectors), and protocol classes [17].

The Unified Process was adopted by Rational (Rational Unified Process - RUP) [19] to be used as an interactive development process (Figure 2). Its models represent the real world, capturing the problem to be solved using the UML-RT notation.

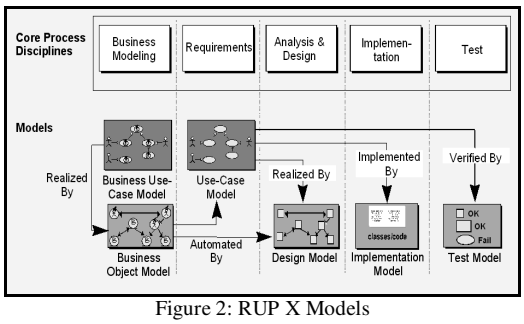

Figure 2: RUP X Models

The combination of customizable code generation and a real-time framework is the key advantage provided by Rational for enabling an MDA approach. MDA enables code generation by using UML-RT models for several embedded designs targeting either a commercial Real-Time Operating Systems (RTOS) or proprietary scheduling environments.

IV. THE FRAMEWORK

Frameworks allow code reuse and fast application developments [2]. Design patterns and frameworks are highly synergetic [9]. While a pattern can be used to describe a framework, a framework can be written as a pattern implementation.

The framework design was based on the QoS requirement of structuring reusable server's structure for Control Station services.

The authors used basic components of RRRT to implement the proposed framework. Besides those basic components, and in order to track and validate the user's requirements the authors created a passive class component to link the use case model to the design model. This assures the effective use case implementation by the framework.

The Control Station (CS) capsule is compound of a Dynamic Service Server (Service Distribution) capsule, Common Protocol Class, and Use Case Class. The structure diagram (Figure 3) was designed to plug-in CS services at run-time, based on external events.

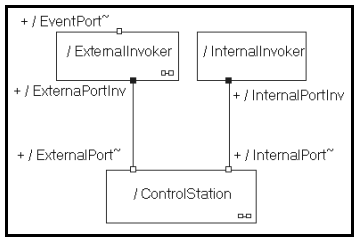

Figure 3. Framework Structure

The External and the Internal Invokers Capsules have specific statechart to implement theirs own behavior. The interoperability design pattern is then applied on their statechart by extending the component state which is responsible for the communication with the CS, during the design phase.

A. Control Station Capsule

In order to implement self-adaptive characteristics, an Event Awareness service, named "Service Distribution" (Figure 4), was created as a Capsule. In this paper, it is considered the transaction of receiving and answering to external event of an Invoker a use case realization. In order to the Service Distribution be able of choosing, at run time, the right service needed by an Invoker, an adaptive policy was implemented.

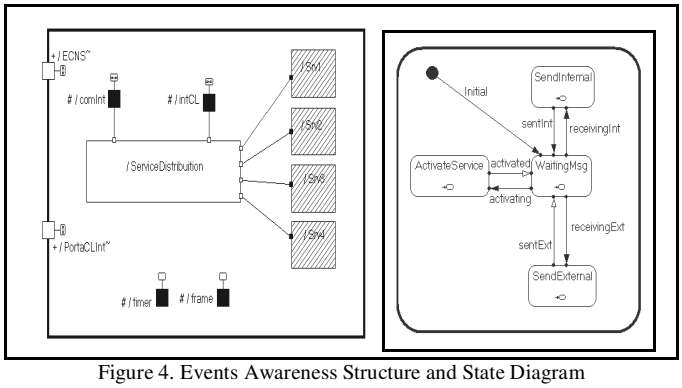

Figure 4. Events Awareness Structure and State Diagram

A self-adaptive policy is defined by a kind of Event-Condition-Action (ECA) rules. The Event is the received message, which is already related to some use case. The Condition which is defined by use cases specifies what service needs to be activated. Finally, the Action is the activation of the correct service. This active component is self-adaptive in the sense of providing the needed service already managed by a use case, which controls events that are received by real world activities. The authors defined the rules to activate Services Capsules based on system use cases that have captured system functional requirements [8].

The ECA rule table (Table 1) is defined during the system design and applied on ActivateService state shown in Figure 4.

Table 1. ECA Rule

Activity /Event of a Invoker	Event message	Use Case	Data	Action of activating a Service
<Activate Service>	<message type>	<UseCase>	<Data>	<Service>

The distribution service component identified in Figure 4 as "ServiceDistribution" implements the communication and data interchange.

B. Service Invoker Capsule

Service Invokers are actors that begin the use cases sequence of action. All Invokers have their own statechart. One of their states performs the communication and data exchange with the CS.

The authors have created their own design patterns as a solution to optimize the interoperability issue into a generalized solution [10]. The pattern is then applied by inserting it in the specific state that exchange data with the CS. Figure 5 shows a basic example of an Invoker Capsule, and the Figure 6 shows the Interoperability Pattern applied to the state responsible to connect to the CS.

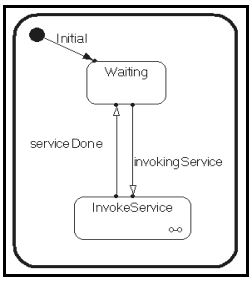

Figure 5. Service Invoker State Diagram

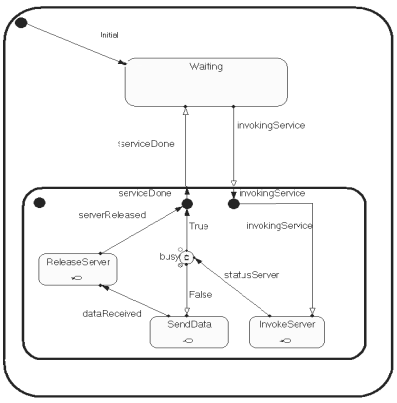

Figure 6. Interoperability Pattern Applied

C. Dynamic Service Server Component

Dynamic Service Servers (Figure 7) are active objects representing the system self-adaptive modeling. Those objects are set as optional in the design phase and are dynamically activated at run-time. This process follows the system adaptive policy written in the rules created to identify and evaluate external events.

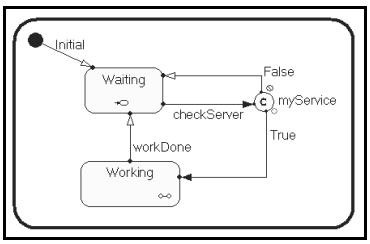

Figure7. Server States Diagram

D. Common Protocol Class

The format design for data interchange was essential for the framework. It has enabled the interoperability among system components by encapsulating it in a structured message class, named Common Protocol Class. This Class contains all needed information to identify the invoker, the service and data transmitted or received (Figure 8).

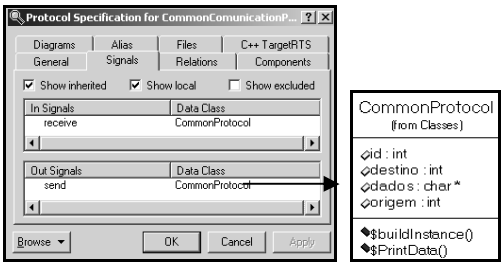

Figure 8. Common Protocol Class

E. Use Case Class

Functional requirements are mapped to the software at the design phase to use cases. The authors defined a passive class, named Use Case Class (Figure 9) to register and trace all use cases that might be realized by the system.

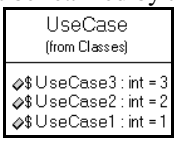

Figure 9. Use Case Class Structure

V. CASE STUDY

The case study uses the proposed framework to create a real-time system prototype to operate an Unmanned Air Vehicle (UAV) from a Control Station (CS) The functionalities for communication, navigation, surveillance, georeferenced information storage, and situational awareness visualization should be designed and implemented (Figure 10).

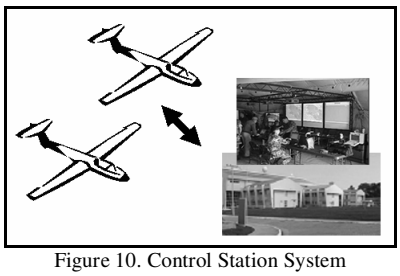

Figure 10. Control Station System

The CS prototype subsystem was created using the Real-time Embedded Systems class [5] at the Brazilian Aeronautics Institute of Technology (*Instituto Tecnológico de Aeronáutica - ITA*). It has been improved, in order to implement self-adaptive characteristics.

The scope of the CS prototype services are:
- Communication UAV-CS (ECOM);
- Storage of Georeferenced Information (EMPM);
- UAV Navigation Control (ENAV);
- Target Remote Sensing (EVIG); and
- Graphic Scenario Visualization (EVIS).

The UAV Route Control Manager (RCM) subsystem prototype, also developed to verify the application of the proposed framework. The RCM main functionality is to control the UAV terminal route within an air-space area to avoid airplane collisions. The RCM invokes the CS to provide the correct UAV route by activating the related services.

A. Adopting the framework

In order to sucessfully reuse the framework, the system analyst must apply the following steps:
Step 1 - Adopting the Communication Protocol Class;
Step 2 - Updating the Use Case Class, by inserting appropriate system's use cases;
Step 3 - Adjusting the system to the framework; and
Step 4 - Expanding the state which communicates to the CS to apply the interoperability pattern.

Figure 11 shows the Use Case Class after the framework adaptation.

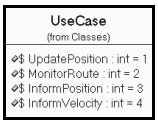

Figure 11. Prototype Use Case Class

Figure 12 shows the prototype structure after the framework adaptation.

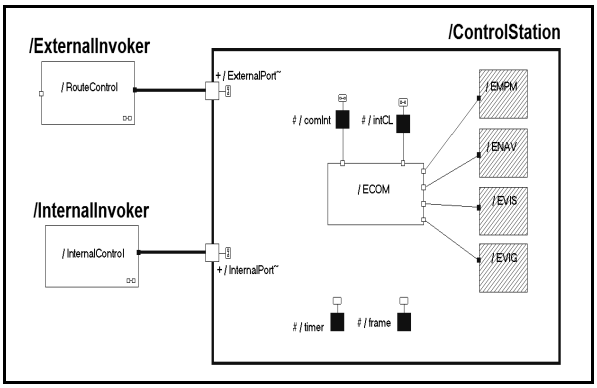

Figure 12. Framework applied to the Prototype Structure

B. Applying the interoperability design pattern

The RouteControl capsule (Figure 13) is the External Invoker of a CS Service. The ConnectCS state has to be expanded to include the interoperability pattern.

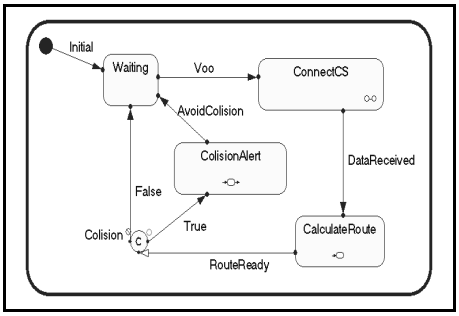

Figure 13. Route Control State Diagram

Expanding the ConnectCS state creates a deeper dimension for the statechart, as shown in Figure 14. The new dimension implements the code reuse concept and allows the new Invoker to instantiate the Interoperability Pattern.

Each state in this new dimension has its own code in C++ following the adaptive rules and keeping track of use cases realizations for the RCM prototype system.

Table 2. Exchanging data for RouteControl

Activity of RouteControl	Event message	Use Case	Data	Service
InvokeServer	requisita	Inform Position	UAV Position	EMPM
SendData	dados	Inform Position	Position Data	EMPM
ReleaseServer	libera	Inform Position	OK	EMPM

ECA rule have already activated the Service, but the code for exchange data have to be inserted in each state of the Interoperability Pattern statechart. The data exchange between the Invoker and the Service is designed on Table 2, which is later translated to the target language (Figure 14).

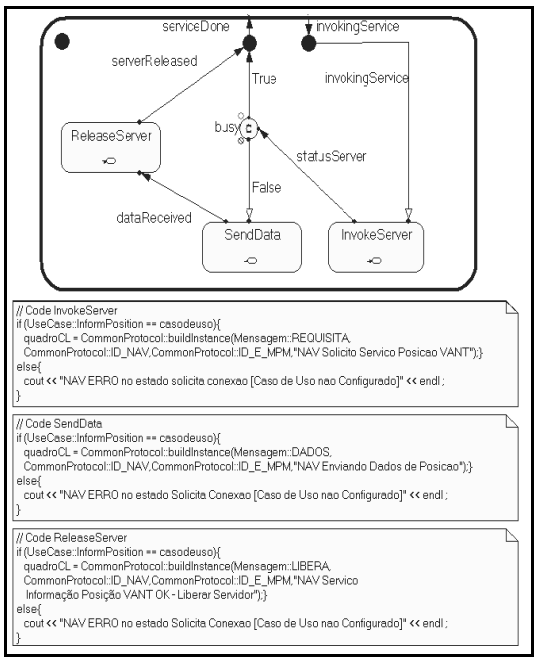

Figure 14. ConnectCS State Expanded

The utilization of self-adaptive mechanisms can be observed at run-time by opening the RouteControl structure diagram (Figure 15). In Figure 16 the prototype execution results are exhibited.

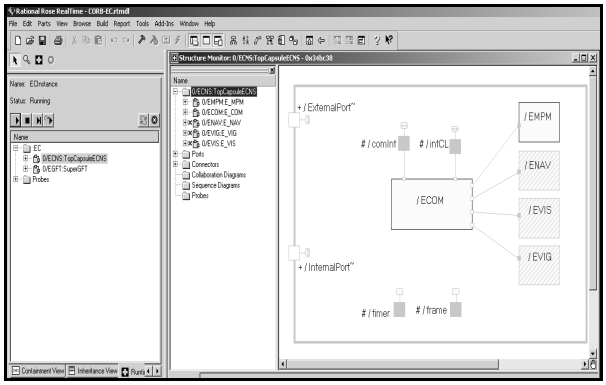

Figure 15. Running the Prototype

```
Rational Rose RealTime C++ Target Run Time System
Release 6.6.2046.0 (+c)
Copyright (c) 1991-2006 Rational Software
rosert: observability listening at tcp port 30482

**********************************************************************
*                Please note: STDIN is turned off.                   *
*  To use the command line, telnet to the above mentioned port.      *
* The _output_ of any command will be displayed in _this_ window.    *
**********************************************************************

EGFT Decolagem Solicitada
EGFT Decolagem Autorizada
EGFT Vant Em Voo
ECOM Recebe Quadro Externo CL
Quadro: 1
        <- 7
        -> 3
        D: NAV Solicito Servico Posicao VANT
EMPM Servico solicitado por: 7
ECOM Recebe Quadro Servidor 3
Quadro: 4
        <- 3
        -> 7
        D: Bem-Vindo ao Servico Mapa Movel
ECOM Recebe Quadro Externo CL
Quadro: 7
        <- 7
        -> 3
        D: NAV Enviando Dados de Posicao
EMPM Dados enviados por: 7
ECOM Recebe Quadro Servidor 3
Quadro: 7
        <- 3
        -> 7
        D: EMPM DADOS RECEBIDOS FORAM PROCESSADOS
ECOM Recebe Quadro Externo CL
Quadro: 2
        <- 7
        -> 3
        D: NAV Servico Informaþõo Posiþõo VANT OK - Liberar Servidor
EMPM Liberando Servico de 7
ECOM Recebe Quadro Servidor 3
Quadro: 6
        <- 3
        -> 7
        D: Servico Mapa Movel Finalizado - Volte Sempre!
EGFT Velocidade e posicao do VANT recebidos
EGFT Rota OK
```

Figure 16. System Results

VI. CONCLUSION

This paper presents the implementation of a self-adaptive component-based framework for real-time system based upon UML-RT and MDA, which a new system can be easily adjusted to the proposed framework.

The creation of a efficient design pattern for interoperability by defining a semantic for message exchange, allowed an easy and transparent message exchange among Invokers and Services of a CS.

The implementation of the self-adaptive mechanism in the framework did not increase the design complexity. Instead, it has allowed a simple solution determination at run-time to the system structure in a given scenario.

A new Invoker was easily integrated to the CS prototype system by applying the Interoperability Design Pattern, due to its structure simplicity and a cost-effective code reuse.

Authors of this paper believe that this approach is unique in the sense of modeling and tracking functional requirements throughout system dimensions.

The major findings of this work are a proposed solution for some gaps of tracking the correct deployment of use cases, and the creation of the interoperability design pattern.

The track of use case realization could be automatically implemented to allow direct transformations of use cases diagrams to statechart. This process can be considered for a future work.

REFERENCES

[1] Booch, G., "Object-oriented analysis and design with applications", Benjamin/Cummings, 1999.

[2] Budd, T., "An Introduction to Object Oriented Programming", Addison Wesley 3rd Ed, 2002.

[3] Colonese, E., and Cunha, A., "An Effective Infrastructure to Develop GIS", In Int. Conference on Information Technology: New Generations, Las Vegas, USA, 2007.

[4] Colonese, E. , "Methodology for Integrating the Scenario Databases of Simulation Systems", Master Thesis AFIT/GCS/ENG/99J-03, USA, 1999.

[5] Cunha, A., "CE-235 Real-time Embedded Systems Lecture Notes", Brazilian Aeronautics Institute of Technology – ITA, 2006. Available: http://www.ita.br/~cunha.

[14] Layaida, O., and Hagimont, D., "Designing Self-Adaptive Multimedia Applications trough Hierarchical Reconfiguration", In Int. Conf. on Distributed Applications and Interoperable Systems (DAIS), Athens, Greece, 2005.

[15] OMG, "Model Driven Architecture", Availabe: http://www .omg.org/mda.

[16] OMG, "UML". Available: http://www.omg.org/gettingstart ed/what_is_uml.htm.

[17] Rational Software Corporation, "DEV470 - Mastering Rational Rose RealTime - Student Material", Volume 1, 2006.

[18] Rational Software Corporation, "Rational Unified Process". Availabe: http://www-128.ibm.com/developerworks/ rational/lib rary/may05/brown/index.html.

[19] Selic, B., Gullekson, G., and Ward, P., "Real-Time Object-Oriented Modeling", John Wiley & Sons, 1994.

[20] Selic, B., and J. Rumbaugh, "Using UML for Modeling Complex Real-Time Systems". Available: http://www-128.ibm. com/developerworks/rational/library/content/03July/1000/1155/115 5_umlmodeling.pdf.

[6] David, P., and Ledoux, T., " Dynamic Adaptation of Non-Functional Concerns", In Int. Workshop on Unanticipated Software Engineering (USE), Malaga, Spain, 2002.

[7] David, P., and Ledoux, T., "Towards a Framework for Self-Adaptive Component-Based Application", Conf. on Distributed Applications and Systems (DAIS), 2003.

[8] Douglass, B., "Capturing Requirements for Real-Time and Embedded Systems", Chief Evangelist, I-Logix, 1998.

[9] Douglass, B., "Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns", Addison Wesley, 1999.

[10] Douglass, B., "Real-time Design Patterns", Addison-Wesley, 2006.

[11] Gamma, E., Helm, R., Johnson, R., Vlissides J, "Design Patterns", Addison-Wesley, 2005.

[12] Hallsteinsen, S. et all, "Self-Adaptation for Everyday Systems", WOSS'04, ACM 1-58113-989-6/04/0010, USA 2004.

[13] Jacobson, I., and Rumbaugh, J. , "The Unified Software Development Process", Addison Wesley Logman, 1999.