AN INTELLIGENT KERNEL FOR THE MAINTENANCE SYSTEM OF A HYDROELECTRIC POWER PLANT

Edgar J. Amaya Simeón, edgar.amaya@gmail.com
Alberto José Álvares, alvares@AlvaresTech.com
Rosimarci Pacheco Tonaco, rosimarci@hotmail.com
Rodrigo de Queiroz Souza, rodrigoqsouza@hotmail.com
Universidade de Brasília, Departamento de Engenharia Mecânica e Mecatrônica, Grupo de Inovação em Automação Industrial (GIAI), CEP 70910-900, Brasília, DF, Brasil

Ricardo Ribeiro Gudwin, gudwin@dca.fee.unicamp.br
Universidade Estadual de Campinas (UNICAMP), Faculdade de Engenharia Elétrica e de Computação, Departamento de Engenharia de Computação e Automação Industrial,Campinas, São Paulo, Brasil.

Abstract. *In this paper, we present the conceptualization of an intelligent system to be used on the predictive maintenance of a hydroelectric power plant. It is a real application, being developed in the context of the project "Modernization Processes of Automation Area of the Hydroelectric power stations of Balbina and Samuel", with the support of Eletronorte. The work described in this paper corresponds to the development of an intelligent kernel, which is planned to be the heart of a major methodology, called SIMPREBAL (Predictive Maintenance System of Balbina), to be implanted in the plant of Balbina, near the city of Manaus in Brazil. The overall specification and design of this intelligent kernel is described here using UML (Unified Modeling Language) diagram. This intelligent kernel, called here I-kernel, will be providing expert systems functionality, besides fuzzy logic rules processing and neural networks in an operational cycle where data is collected from equipments and databases, and will be used to both give support to the operational team and to the maintenance team at the plant. I-Kernel will get the data from supervision system by means of both JDBC access to databases and JNI (Java Native Interface) getting data from the OPC (OLE - Object Linking and Embedding - for Process Control) server. In order to provide expert systems and fuzzy logic functionality to the system, we will be using the tools JESS (Java Expert Systems Shell) and Fuzzy-JESS. The SIMPREBAL methodology is based on RCM (Reliability centered maintenance) concepts, being used to analyze the manners and effects of fails on the Hydraulic Generator Units of Balbina, focusing its analysis in the Turbine system. In this work, the main details of the I-Kernel system are detailed. The concept of an intelligent kernel is that will be useful for the construction of similar applications in the future.*

Keywords: *Unified Modeling Language, Predictive Maintenance, Expert Systems, Fieldbus Foundation.*

1. INTRODUCTION

The motivation for this work started with the development of the SIMPREBAL (Alvares, 2006) methodology, under the context of the project "Modernization Processes of Automation Area of the Hydroelectric power stations of Balbina and Samuel", with the support of Eletronorte. The main goal was to implement ideas from RCM (Reliability Centered Maintenance) (Smith, 1993)(Mobray, 1997) and implant them in the plant of Balbina, a hydroelectric power plant located near the city of Manaus in Brazil. The SIMPREBAL project conceived an intelligent system collecting data directly from equipments in the power plant, storing this data in a database, and making use of all this information in order to predict maintenance interventions in order to minimize the impact of (future) failures in the operation of the plant. The main strategy for achieving this goal is by means of the prediction of future failures, based on the measuring of some process variables and analyzing their operational conditions in order to detect anomalous situations that may indicate that a potential failure is possible, in a near future. This may allow for the maintenance team to anticipate this failure and take the appropriate actions in order to minimize the problems that this future failure may impose in the operation of the power plant. In order to develop the SIMPREBAL methodology, there was a need for an intelligent kernel, a software component responsible for getting the data both directly from the equipments used in the power plant, and from databases with historical data, processing all this data in order to analyze them, making predictions, and an integration with the maintenance system of the power plant, so the predicted anomalies may take their role in the organizational process of the maintenance team. From this demand, there appeared the main subject theme of this paper, the I-Kernel Subsystem. We conceived I-Kernel to be a software component, to be developed in Java language, which should be integrated to the main computer system in Balbina, providing the way for implementing SIMPREBAL in the operational (supervision and control) and maintenance organizational process already working in the plant of Balbina.

The role of I-Kernel was, then, to provide the general functionalities of expert system, fuzzy rule based system and neural networks, which were required in order to generate the predictive analysis and decision-making demanded by the SIMPREBAL methodology. In this paper, we detail how we conceived this main component and how it can be integrated to other sub-systems in order to allow for the SIMPREBAL methodology to be applied in a power plant. This will be

developed in the next sections, by means of a detailed object-oriented specification and design.

The OO (object-oriented) paradigm offers several benefits, such as encapsulation, abstraction, and reusability to improve the quality of software. The UML (Unified Modeling Language) has emerged as the standard for analysis and design of OO systems. UML provides a variety of diagramming notations for capturing design information from different perspectives. In recent years, researchers have realized the potential of UML models as a source of information in software development (Sommerville 2003).

The rest of this paper is organized as follows: Section 2 presents the I-kernel subsystem requirements as one of the main subsystems of the SIMPREBAL system; Section 3 describes the general SIMPREBAL system configuration, showing how the I-Kernel should be integrated to other sub-systems; Section 4 presents the design of the I-kernel application ; Section 5 presents the Configuration and Monitoring Application Tool that works together with the I-kernel aplication; and finally Section 6 concludes the paper.

2. I-KERNEL SUBSYSTEM SPECIFICATION

The I-Kernel subsystem is a software component, to be developed in Java language, responsible for the data acquisition from databases and process equipments through OPC, and its processing through conventional rules, fuzzy logic and neural network, in N layers. As much of the functionalities required for I-Kernel are already available as out-of-thebox software components, instead of developing the whole system from scratch, our proposal is to use the available components and to integrate them in order to generate I-Kernel. In the same way, I-Kernel, as a software component, may be reused in future projects that demand intelligent processing solutions involving the processing of rules, fuzzy logic or neural networks.

The components to be used are the following: JESS - Java Expert System Shell, developed by Sandia National Laboratories, responsible by the rule processing and main inference machine to the expert system part of I-Kernel; Fuzzy-JESS, developed by the NRC Institute for Information Technology, responsible for the processing of fuzzy rules, in a way similar to JESS; JDBC responsible for the access to the databases and JNI, responsible for the access to the OPC servers which give direct access to the equipments. The I-Kernel application needs to access different databases, in order to integrate to the computational environment of Balbina's power plant. This includes the database for the Rockwell's Monitoring System, Smar's Asset Management System - Assetview, IBM/MRO's Operating and Maintenance Management System MAXIMO-MES(Manufacturing Execution System), and also the own SIMPREBAL database which will be storing all the knowledge required by the different intelligent algorithms. The Neural Network module will be developed almost from scratch, based on some code available at UNICAMP, built according to the scientific literature. A general view of this architecture, in terms of a UML diagram, is available at Fig. 1.

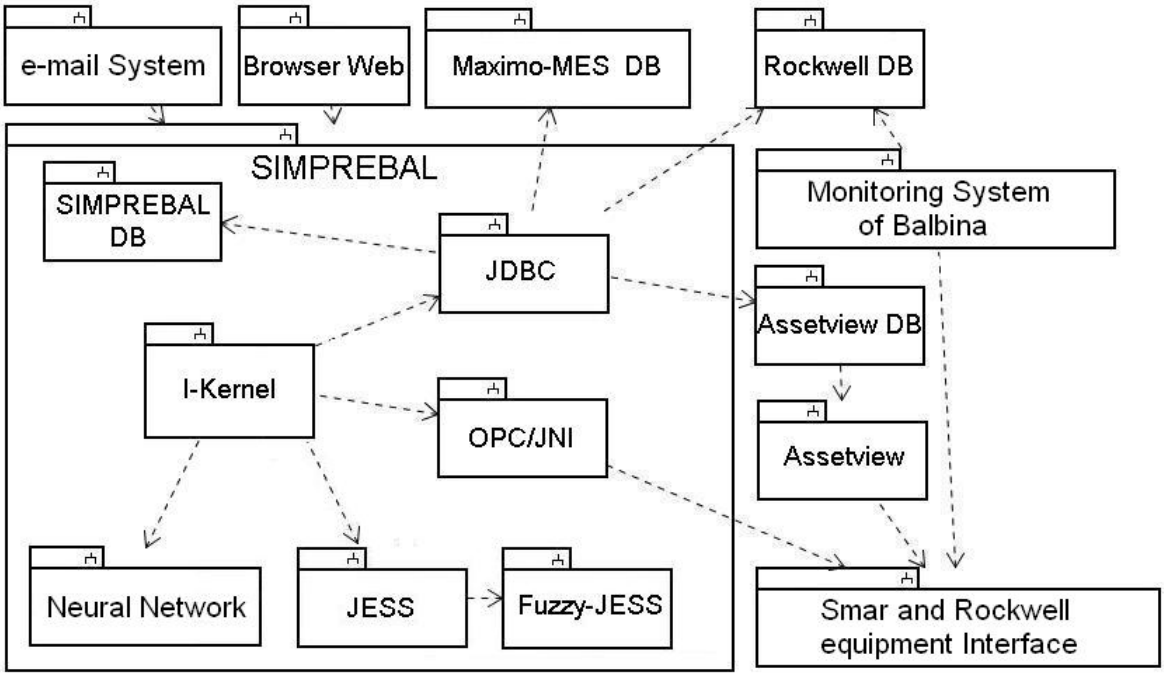

Figure 1. SIMPREBAL System Architecture and Interactions

According to (Sommerville, 2003), a useful way to structure the requirements is to split them into user requirements and system requirements.

2.1 User Requirements

The user requirements should capture the more general demands of system's users. Normally, these requirements are described in natural language and shouldn't enter too much in technical details. In this document we will be adopting this format recommended by Sommerville, listing the user requirements as a requirements list in natural language. The user requirements are divided into "Functional Requirements" as illustrated in Table 1, and Non-Functional Requirements listed in Table 2.

Table 1. Functional Requirements.

	Functional Requirement
FU1	The system should access the Plant's data, either from databases used by the Monitoring System of Balbina, or directly from the plant's equipments, by means of an OPC server, which should make available the on-line information from the equipments
FU2	The system should process those data in the form of: a rule-based expert system, a fuzzy logic inference engine, Neural Networks
FU3	The system should alert the user through e-mail messages when anomalous situations could be diagnosed
FU4	The system should alert the user through a visual alert, when anomalous situations could be diagnosed
FU5	The system should permit the edition of a synoptic, containing some variables to be monitored, chosen by the user and composing a possible presentation screen
FU6	The system should exhibit the on-line value of the variables being monitored, which were selected to compose a certain synoptic, showing them in a proper screen previously developed
FU7	The system should implement some learning mechanism, in such a way that the report of failures and previous defects could be used to prevent the appearance of new failures
FU8	The information processing should happen as a closed operational cycle, that should follow the following sequence: Verification of the Data to be acquired; Acquisition of the Data from Database; Acquisition of the Data through OPC; Temporary storage of all the data in JESS variables; [For each one of N possible layers: Processing of the rules through JESS, the Fuzzy Logic rules through Fuzzy-JESS, the data through Neural Network;] Updating of the Data in the Database; Updating of the Data through OPC.

Table 2. Not-Functional Requirements.

	Not-Functional Requirement
NFU1	The system should be developed in Java language
NFU2	The system rules should not be stored directly in the source-code, but should be editable and available externally in an editable text file
NFU3	The system should possess a web interface access, through which should be possible to the user to configure the system, to edit rules and parameters and to monitor the variables being processed by the system
NFU4	The system should be compatible to SQL generic databases, given that a JDBC driver for the database is available
NFU5	The system should use the JESS package as an inference engine
NFU6	The system should use the Fuzzy-JESS package to process the fuzzy logic rules.
NFU7	The system should be conceived such as classic rules, fuzzy logic rules and neural network may be used in an interchangeable way for each one of the SIMPREBAL processing layers (3-condition monitor, 4-health assessment, 5-prognostics, 6-decision support) according to the OSA-CBM (Open System Architecture for Condition Based Maintenance)

2.2 System Requirements

The system requirements will be detailed in a more technical way, using UML diagram. Following the Unified Development Process (Jacobson et.al. 1999), system requirements should be specified by means of use cases. In Fig. 2, we have an UML use-case diagram, which provides us a general overview of the system. The system is composed of two applications that are executed in an independent way: the I-Kernel application and the Configuration and Monitoring Application (Gudwin 2006).

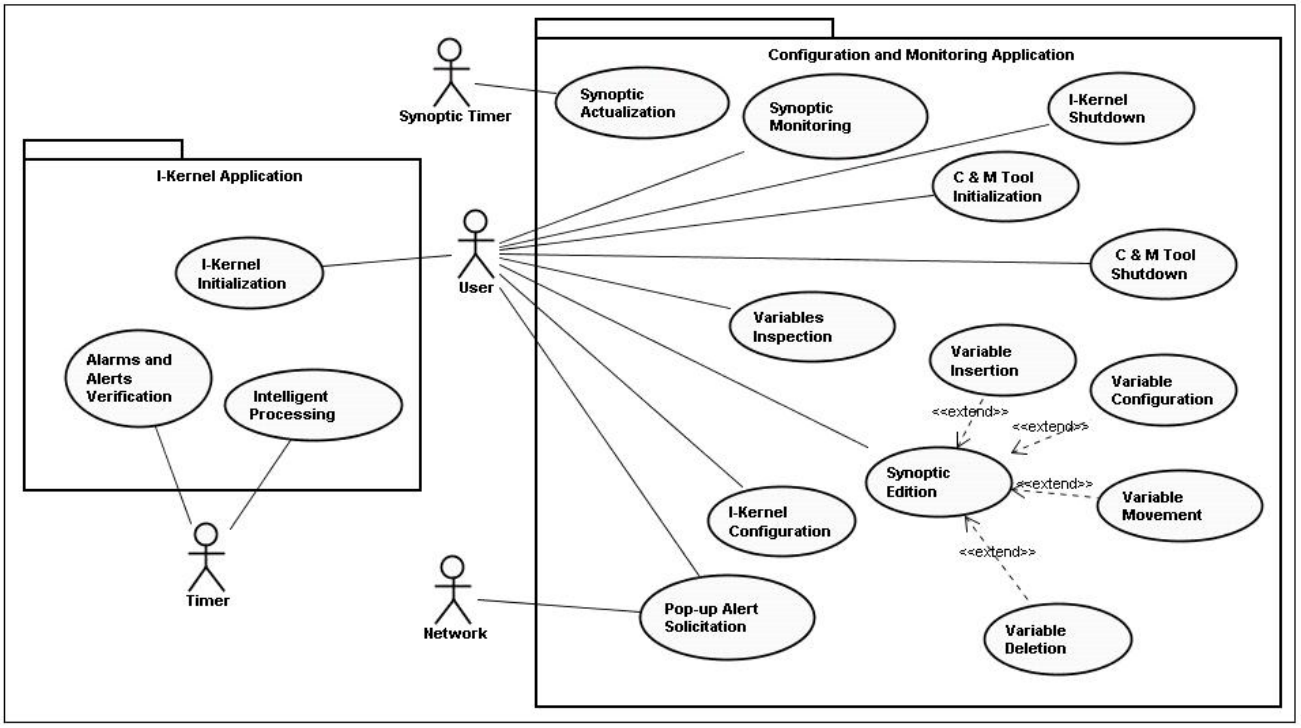

Figure 2. SIMPREBAL Use-Case Diagram

The overall use-case diagram represents an overview of the system's functionality by pointing out who are the main users of the system and to which use-cases they take part. The whole system is partitioned in two sub-systems: The I-Kernel Application and the Configuration and Monitoring Application. The I-Kernel Application is responsible for the main operational cycle, where data is collected, processed and dispatched. The Configuration and Monitoring Application, as its name suggests, is responsible for configuring and monitoring I-Kernel.

We developed basically 16 use-cases: I-Kernel Initialization, Intelligent Processing, Alarms and Alerts Verification, I-Kernel Shutdown, C & M Tool Initialization, C & M Tool Shutdown, Synoptic Edition, Variable Insertion, Variable Deletion, Variable Movement, Variable Configuration, Synoptic Monitoring, Synoptic Actualization, Variables Inspection, I-Kernel Configuration, and Pop-up Alert Solicitation.

3. System Configuration

In Fig. 3 we present a UML deployment diagram, showing a possible configuration for the final system. This is not the only possible configuration, but it is a prototypical one. In the case of Fig. 3, there might be 1 client machine and 3 different server computers, where the different software modules should be installed. A different configuration might use only one server computer, where all the service modules are installed. The only restriction presented is that the ConfMonitToolApp module and the I-kernelApp module necessarily need to be installed in the same computer. This restriction is because the ConfMonitToolApp module is a java applet, which uses the network for communicate to the I-kernelApp module. For security restrictions, a java applet only can communicate with the same computer from where it was loaded. For that reason, it is necessary that the I-kernelApp module is installed in the same computer where the web server is installed. Besides the I-kernelApp and the ConfMonitToolApp, other services to be required are: a web server, a database compatible to JDBC and an OPC server to be accessed through JNI. The two software components which were developed were the I-kernelApp and ConfMonitToolApp. These applications should be available as JAR files which should be executed independently through a JVM (Java Virtual Machine). The I-KernelApp module is a standalone java application. The ConfMonitToolApp is a java applet, loaded from a website, and stored in the web server.

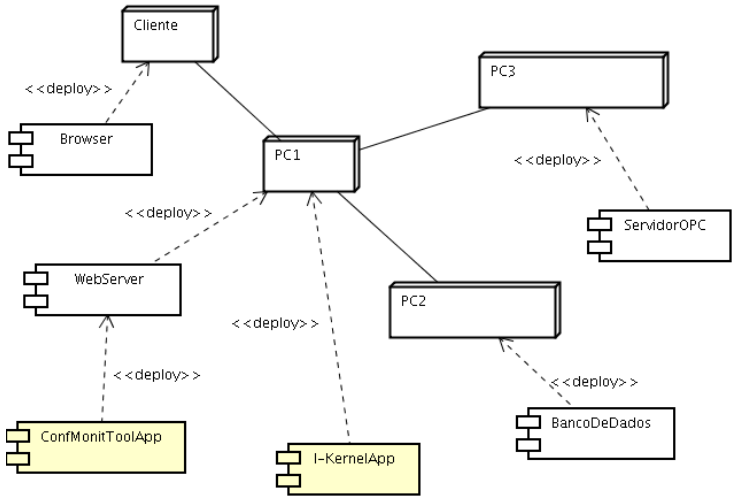

Figure 3. System Configuration

4. I-Kernel Application

The I-Kernel application is a server that doesn't have an user interface. In this way, the only thing that the user can do is to start the application. After that, a timer previously programmed sends periodic "ticks" that start each operational cycle of intelligent processing and make alarms and alerts verification. The details can be observed in Fig. 4.

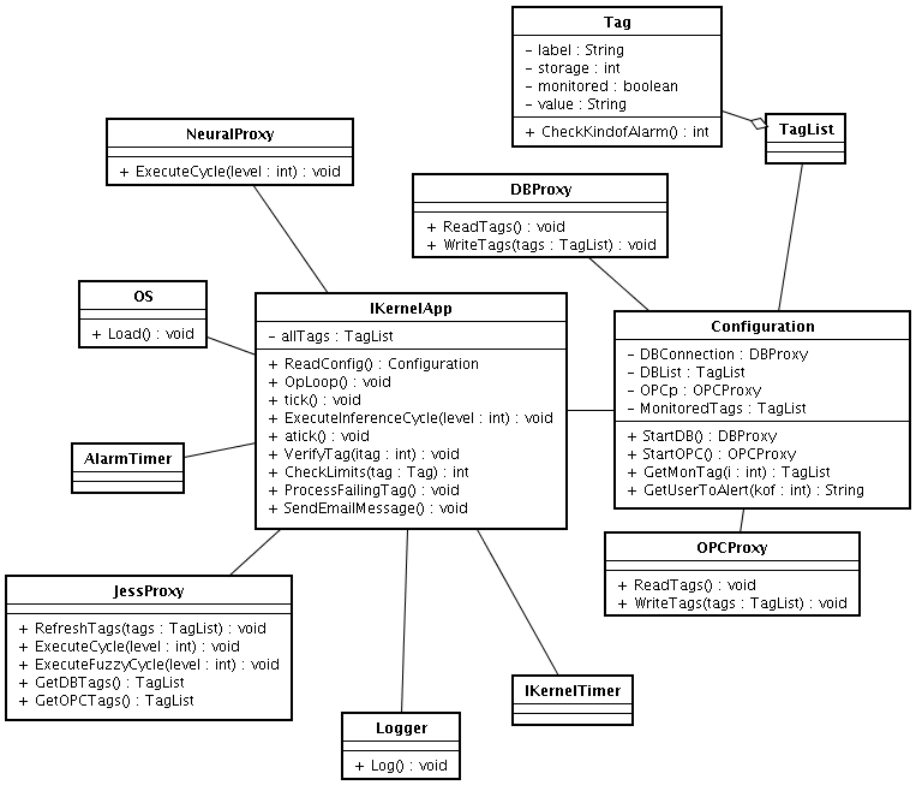

Figure 4. I-Kernel Application Use-Case Diagram

4.1 Use-Case 1: I-Kernel Initialization

In order to initialize the I-kernel application, the user asks the operational system to load the application. Once the application is loaded, it creates an IKernelApp object, which reads the configuration file, creates a new AlarmTimer, a new IKernelTimer, ask for the starting of the database proxy and to the OPC proxy, and after that creates the JESS proxy

(which should include the Fuzzy-JESS processing) and creates the neural networks proxy. Once created, the AlarmTimer and the IKernelTimers start doing "ticks", which will trigger the main operational cycle. This sequence is shown in Fig. 5.

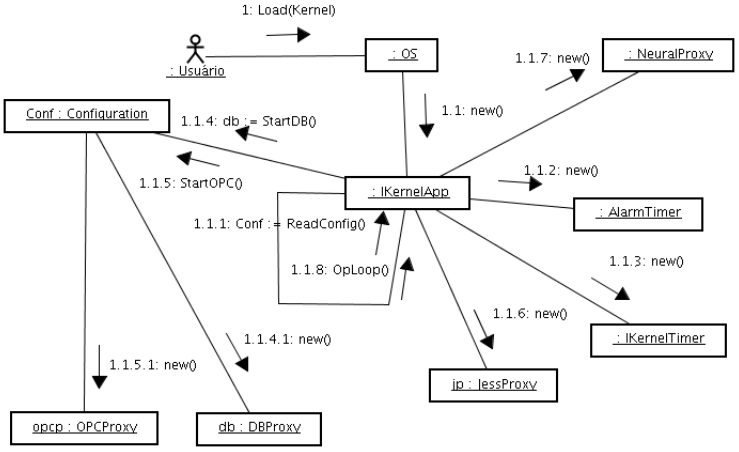

Figure 5. I-Kernel Application Initialization Collaboration Diagram

4.2 Use-Case 2: Intelligent Processing

The intelligent processing is the heart of the application. It starts with a "tick" from the IKernelTimer, which trigger a sequence of operations: first it asks the DBProxy for the tags to be read from the database, then it asks the OPCProxy for the tags to be read directly from the Process Equipments, then bot tags from database and from OPC servers are fed into JESS. After that, the system starts an n-level loop, where for each loop, the system asks the JESS engine to perform one operational cycle, which first execute JESS rules, then Fuzzy JESS rules and then the neural network. The whole process is shown in a collaboration diagram in Fig. 6.

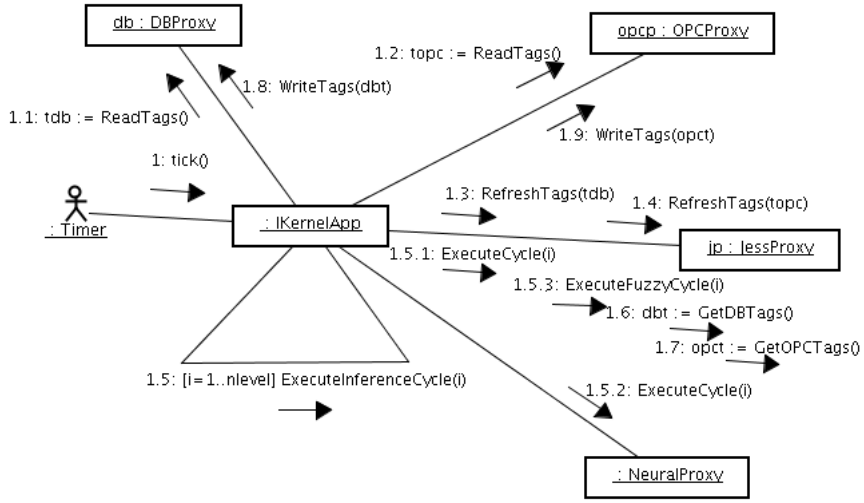

Figure 6. Intelligent Processing Collaboration Diagram

4.3 Use-Case 3: Alarms and Alerts Verification

The alarms and alerts sub-system works in parallell to the intelligent processing, and the mechanism is somewhat the same. The AlarmTimer perform periodic "ticks" to the IKernelApp, which in response perform a verification of alarms and alerts. The system verify a certain number of monitored tags, and for each tag it checks if the current value is out of bounds. If the tag value is out of normal, the system checks if the monitored tag should send an e-mail alert or a popup to the operator, as defined in configuration. After that, it logs the fact with the Logger object and performs either the sending of an e-mail message or a popup alert message. This behavior is described in Fig. 7.

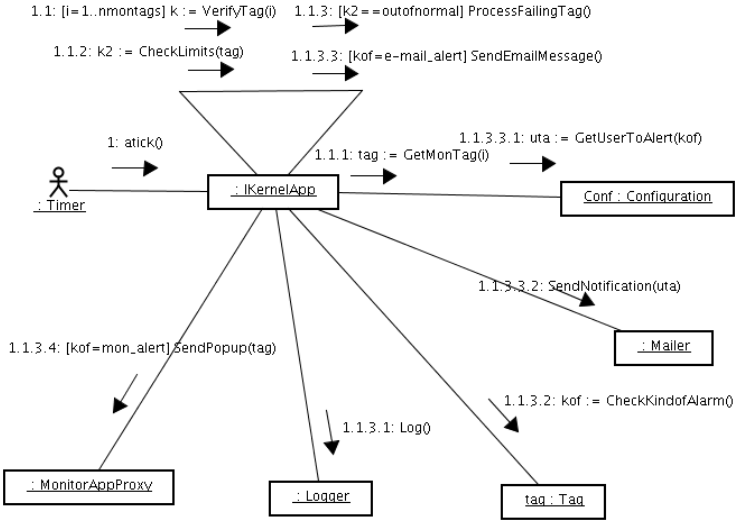

Figure 7. Alarms and Alerts Verification Collaboration Diagram

5. Configuration and Monitoring Application

The Configuration and Monitoring application is a web application, destined to promote the configuration of parameters of the I-Kernel, as well as the Monitoring of the variables under control of the system and the request of shutdown of the I-Kernel. This monitoring can be a direct supervision of the state of some variable (by the selection of the variable among all those which are available), or it can be a monitoring by synoptic. In the direct variables monitoring, the user chooses the variable to be monitored, and the system exhibits its state directly.

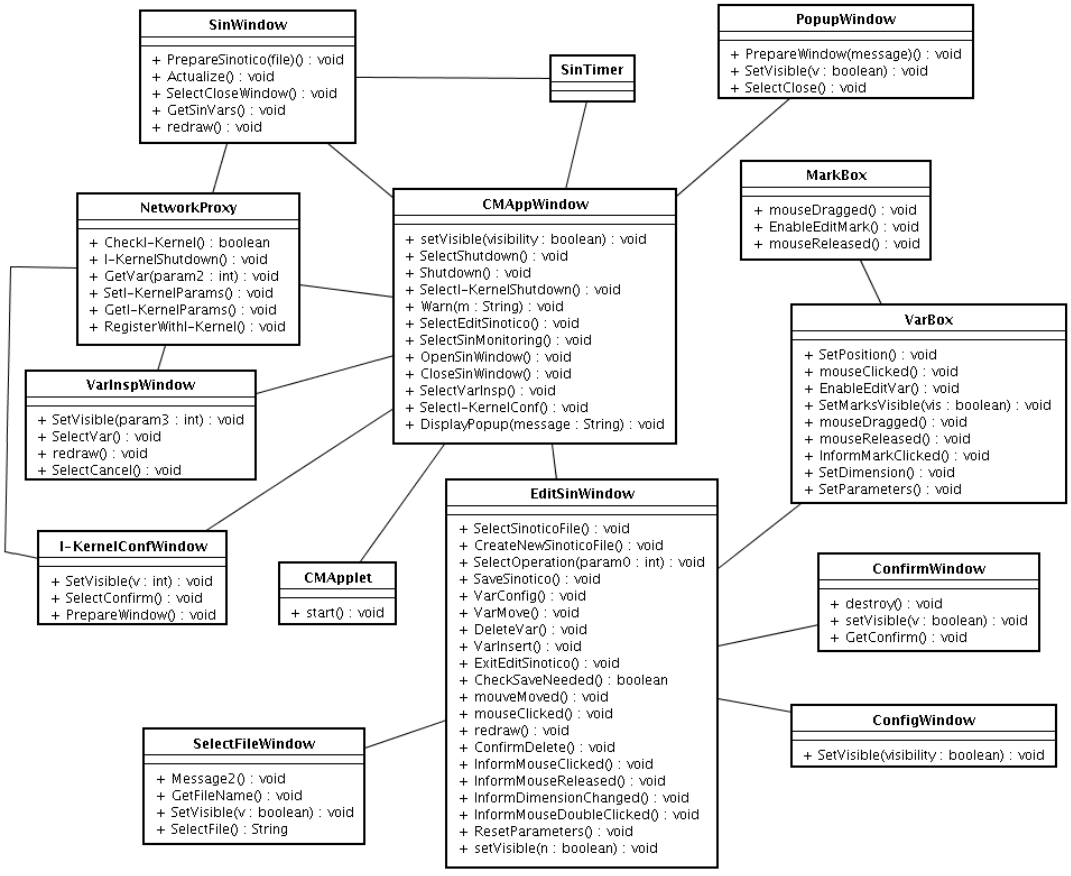

Figure 8. Configuration and Monitoring Application Collaboration Diagram

This monitoring type is targeted to an isolated inspection in the state of some variable of the system. In the synoptic monitoring, it's first required the definition of a synoptic, through the configuration of a screen with a group of variables that one want to monitor, and once a synoptic has been created, the user can choose it for synoptic monitoring. Besides these use cases, the system must treat request from the network, usually originated by the I-Kernel application, so that an alarm and alert pop-up is exhibited for the user. That pop-up request is usually generated from the verification of alarms and alerts use case at the I-Kernel. All the parameters and classes are shown in Fig. 8.

5.1 Use-Case 4: I-Kernel Shutdown

For this use case to start, the C & M tool should be active. To shutdown the I-Kernel application, the user first asks the I-kernel to shutdown. The system then verifies the I-kernel status. If it's off, then it sends an error message, but if it's on, the system starts the shutdown of the active I-Kernel.

5.2 Use-Case 5: C & M Tool Initialization

The user loads an URL from the Web browser, which corresponds to the address of the Configuration and Monitoring Tool. The web page loads the CMApplet which should initialize the C & M tool and open its main user interaction window.

5.3 Use-Case 6: C & M Tool Shutdown

Once the C & M Tool is open and operational, the user requests the system to close this tool.

5.4 Use-Case 7: Synoptic Edition

This is a complex use-case, which is extended by 4 other use-cases. Through this use-case, the user edits a synoptic to be used in another use case: Synoptic Monitoring. Basically, the user informs the synoptic name to be edited (it can be a new synoptic or a synoptic previously edited), and the system opens that synoptic for edition. With the synoptic opened, the user may have a series of options that can be chosen in any order: Variable Insertion, Variable Deletion, Variable Movement, Variable Configuration, Save synoptic, and conclude synoptic edition. If the user made all of the necessary editions, he can decide to save or close the synoptic. The details of this use case are in Fig. 9.

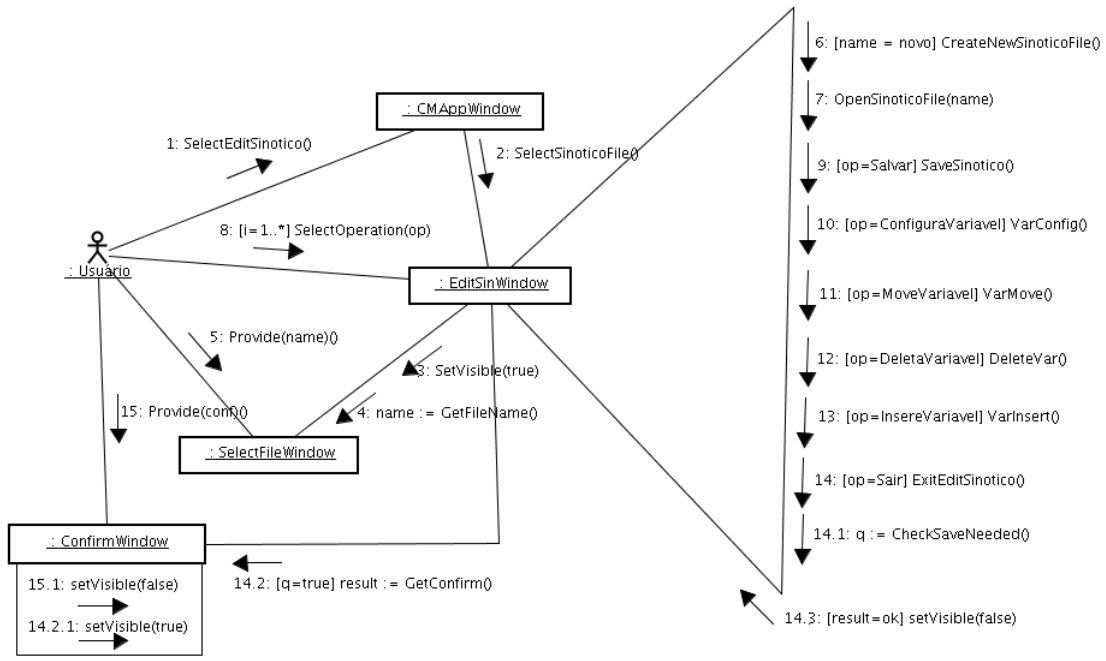

Figure 9. Synoptic Edition Collaboration Diagram

5.5 Use-Case 8: Variable Insertion

This use-case is an extension of the "Synoptic Edition" use-case, allowing the user to insert a new variable to be monitored in the synoptic being edited. Basically the user selects the new variable to insert, positioning the mouse where he wants it to be inserted. The system inserts the new variable being monitored and upgrades the screen.

5.6 Use-Case 9: Variable Deletion

This use-case is an extension of the "Synoptic Edition" use-case, allowing the user delete a variable being monitored in the synoptic. Basically, the user selects the variable to be deleted and asks the system to delete it.

5.7 Use-Case 10: Variable Movement

This use-case is an extension of the "Synoptic Edition" use-case, allowing the user to move around a variable being monitored in the synoptic. Basically, the user clicks on the variable to mark it. After that he clicks in it again, without releasing the mouse button and then moves it for the final position. When releasing the mouse button, the system moves the variable to the wanted position.

5.8 Use-Case 11: Variable Configuration

Permit the user to configure the variable monitored in the synoptic edition. Basically the user has two options: he can either resize the variable, or modify the variable's configuration parameters. If he wants to resize the variable, he clicks in one of the variable marks and moves it to the new wanted point. The system will resize the variable considering the new point. Alternatively, the user can give a double click in the variable, and a new window with the configuration parametes will appear. In it, the user can edit all the variable parameters which he wants, and after finishing the configuration, he can confirm or cancel the alterations.

5.9 Use-Case 12: Synoptic Monitoring

After a synoptic is edited in use case 7: Synoptic Edition, it may be used by the user to obtain the monitoring of a group of variables in a window, defined in the synoptic template. The user basically chooses the file with the synoptic template, and the system opens the monitoring window and it begins the timer that will make the updating of the screen. After that, the system will be waiting to the user request for closing the window, and when this happens, it deletes the timer and the monitoring window.

5.10 Use-Case 13: Synoptic Actualization

This use-case complements the use-case 12: Synoptic Monitoring. Basically, when a synoptic window is being monitored, it is associated with a synoptic timer that sends periodic "ticks" of clock. In each "tick", the system captures the synoptic variables through OPC server and database (it has been configured through the use-case Variable Configuration), updates its values and update the screen to see the new values.

5.11 Use-Case 14: Variables Inspection

The user has the option of inspecting one or more variables. The user selects the variables to inspection, and the system opens a variables inspection window. The user can select which variable he wants to inspect, and the system show that variable in the appropriate place and exhibits the variable value. The user then can decide to inspect another variable or to conclude the operation.

5.12 Use-Case 15: I-Kernel Configuration

The user can update the parameters of the I-Kernel configuration. The user requests to the system the I-Kernel configuration, and the system opens a configuration window, containing all of the parameters configured. The user then can make the alterations that he intends, and confirm the changes or to leave canceling the operation.

5.13 Use-Case 16: Pop-up Alert Solicitation

This use-case is used together with use-case 3: Alarms and Alerts Verification. The I-Kernel subsystem sends a request through the network asking for the alert pop-up to appear. In the present use-case, the request arrives through the network, and the C & M subsystem creates the pop-up window, showing the message wanted to the user. After reading the message the user can request the closing of the pop-up window.

6. CONCLUSIONS

We briefly presented in this paper the requirements and design of an intelligent kernel (I-Kernel) to be used in the construction of a Predictive Maintenance System of a Power Plant. The modelling complies with RCM (Reliability centered maintenance) concepts. The system basically consist of the ConfMonitToolApp module and the I-kernelApp module. This application is actually going to a coding phase, and the next steps are to fully test the application within the power plant. Despite its use in this particular application, the concept of an intelligent kernel goes beyond simply this project, being useful for the construction of similar applications in the future.

7. ACKNOWLEDGEMENTS

We acknowledge the support of the Eletronorte and Manaus Energia provided by the Reserch and Development Program under contract number 4500052325, project number 128 "Modernization of Processes Automation Area of the Hydroelectric power stations of Balbina and Samuel", that has as a technical responsible Prof. Alberto José Álvares of the UnB, and we acknowledge too to the engineer Antonio Araujo from Eletronorte, who played a significant role in this work for the information provided that was import to develop this project.

8. REFERENCES

Alvares Alberto, 2006, "SIMPREBAL: Metodologia do Sistema de Manutenção Preditiva da Usina de Balbina Baseado dos dados Monitorados do Sistema De Supervisão e Controle Smar e Rockwell", Relatório Técnico de Pesquisa UnB.

Gudwin Ricardo, 2006, "Especificação do Sistema - Sistema I-Kernel: Um Kernel Inteligente para o SIMPREBAL - Sistema de Manutenção Preditiva de Balbina", Relatório Técnico.

Jacobson Ivar, Grady Booch, James Rumbaugh, 1999, "The Unified Software Development Process", Addisson Wesley.

Moubray John,1997, "RCM II - Reliability-Centered Maintenance" 2nd edition, New York: Industrial Press Inc.

Smith Anthony M., 1993, "Reliability-Centered Maintenance" - McGraw-Hill, USA.

Sommerville Ian, "Engenharia de Software", 6a. edição, Addison-Wesley/Pearson