A Real-Time Web-based Graphic Display System using Java™ LiveConnect Technology for the Laguna Verde Nuclear Power Plant

Efren Ruben Coronel Flores, Ilse Leal Aulenbacher

***Abstract*—**This paper describes the architecture of a real-time Web application, which is being developed for the new monitoring system in the Laguna Verde Nuclear Power Plant. We describe technologies and methodologies that were applied to achieve a correct implementation. We also describe the main technological challenges that were faced in order to develop a Web-based real-time application. The application was developed using Java and JavaScript, making use of LiveConnect technology to achieve interoperability between both languages. HTML5, CSS3 and SVG were used to create graphic user interface of the application.

***Keywords*—**Real-Time, Nuclear Power Plant, Web Application, Process Information.

I. Introduction

GRAPHIC displays present relevant information regarding operating conditions in power plants. This information is of vital importance to operators, because it aids in decision-making and is needed for power plant operation. Therefore, graphic displays need to be highly reliable and information must be presented clearly and concisely. In the particular case of nuclear power plants, graphic displays are often subject to a series of regulations that have to do with how information is presented to operators.

This paper describes the architecture and design of a Web application that will replace the current graphic display system in the Laguna Verde Nuclear Power Plant. One of the main requirements for this application is that it needs to be integrated to an existing real-time data acquisition system (DAS). In addition, the application must meet strict performance and reliability criteria since it is a mission-critical system that must be able to operate for extended periods of time with no interruptions. Furthermore, information presented to operators must be highly precise and reliable. Another important requirement is to ensure that the application can be supported for at least 15 years. For this reason, the application must be built using well-established languages and standards. Since the power plant is under constant renovation and maintenance, graphic displays should be easy to update in a reliable manner. Finally, processing should be distributed to client workstations in order to minimize the load in DAS servers.

Taking into consideration the application requirements stated above, we decided to develop the new graphic display system as a Web-based application. Currently, Web-based applications are becoming increasingly popular and technologies used to develop them are open and well-established; thus, software lifespan for this type of applications can be expected to be considerable. These technologies also offer a higher ease of use with respect to other programming languages. Contrary to what might be expected, we show that it is possible to develop a mission-critical, Web-based application that works in real-time.

This paper first gives a brief overview of the data acquisition system to which we will integrate our graphic display system and that will be the main data source for our graphic displays. We then present the proposed architecture, explain its modules and describe how we tackled several technological challenges to achieve a high degree of performance and reliability. Finally, we explain our graphic user interface implementation.

II. Background

The Laguna Verde Nuclear Power Plant relies on an information system known as SIIP, to monitor its processes. The system consists of servers which acquire store and process data in real time. It also consists of workstations with graphic displays, which present real-time information on key processes and systems. The SIIP system is mission-critical and is integrated with a data acquisition system (DAS) known as NSAD [1], which was developed by the Electrical Research Institute of Mexico. The NSAD system can acquire data from different sources such as RTP [2], NUMAC (Nuclear Measurement Analysis and Control) and DEHC [3] (Digital Electro-Hydraulic Control) modules, among others. This system also features software modules capable of generating composed data points, which are complex data points that calculate important parameters through specialized algorithms. Such calculations include operational limits, balance of plant, security parameters, etc. The NSAD system is also capable of generating long duration historical archives known as SCAN [4], which are used to analyze transients or important events as well as to generate tabular or graphic trend reports

III. Architecture

Based on the requirements, we analyzed and evaluated two different possible architectures: desktop and Web-based applications. Each option has its advantages and disadvantages. For instance, desktop applications are robust and can be developed in a variety of programming languages. However, these applications may become dependent on the platform in which they were developed. Regarding the development of visual displays with complex drawings, desktop applications require higher development efforts due to the lack of an easy-to-use drawing standard. Therefore, in most cases, it is necessary to resort to third-party applications.

Web applications have their disadvantages as well, which are mostly related to communication or reliability in the context of mission-critical real-time systems. However, these applications have many advantages. For instance, Web applications are widely used and thus, extensive support in the future is expected. This is mainly because they are based on easy-to-use, open and well-established standards. Another advantage is platform independence; to deploy these applications, a Web browser with support for the required technologies is all that is needed.

Fig. 1 illustrates the proposed architecture, which highlights and defines three main modules: communication, processing and presentation. Each module makes use of different technologies. Therefore, our application requires a Web browser that supports Java, JavaScript, HTML5 [6], CSS3 [7] and SVG [8]. The latter is used for vectorial drawings in graphic displays.

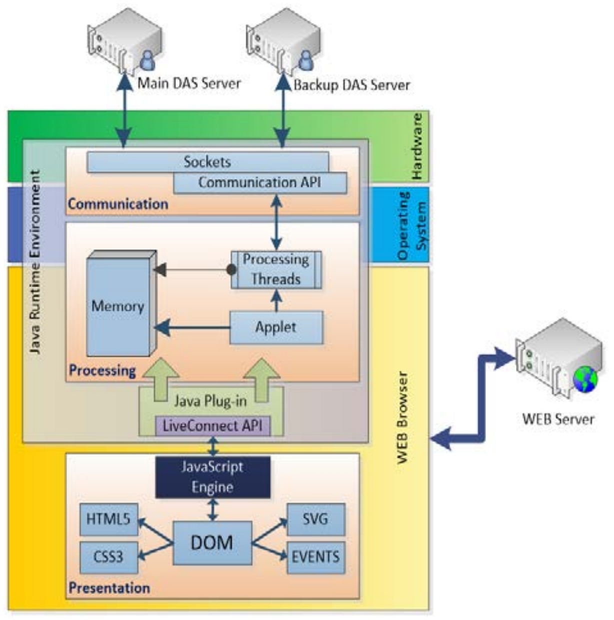

Fig. 1 General Architecture

When a Web browser loads the application, it creates a Java applet which in turn launches the Java execution environment, which is better known as the Java Virtual Machine (JVM). It is in the JVM where Java objects for both processing and communication modules are created. The JVM allows creating multiple execution threads that run asynchronously in relation to the main browser thread. From within these threads, it is possible to use sockets to establish communication with the DAS server to exchange information. Multi-threading also allows the application to perform advanced processing tasks, which require greater processing time and resources. Such complex tasks would cause performance problems if they were assigned to the main browser thread.

Finally, within the presentation module, JavaScript is used to manipulate and interact in real-time with the browser visualization environment. This is done through LiveConnect technology [9], which enables interoperability between the JavaScript engine and Java applets; which in turn, allow obtaining information stored in the JVM execution threads. LiveConnect technology is implemented in every Web browser and allows Java applets to communicate with the JavaScript engine and vice versa. Furthermore, it is in the presentation module where open standards dedicated to the visual part of the application converge: HTML5 is used to define elements such as texts, tables, labels, buttons, etc. and SVG is used to draw complex elements such as the representation of a turbine, a condenser, or a nuclear reactor vase.

IV. Communication

Communication is the main building block in real-time applications, since a high degree of performance and reliability is required. Hence, it is one of the most important requirements in our Web-based graphic display system. To meet such requirements, the main challenge was to determine which technology would be more adequate. One important consideration is that to obtain a high level of performance, the application needs to have dedicated point-to-point communication; this is achieved by using socket libraries over a well-defined protocol. Most programming languages designed for traditional desktop applications, implement sockets and their use is relatively simple. However, it is in Web environments where the use of sockets becomes more complicated.

Recently, new technologies have emerged in order to solve this problem; one of them is the Websockets standard [5], which is expected to bring real-time or close-to-real-time communication capabilities to Web environments. Websockets provide full-duplex communication over a TCP connection. There are libraries that already make use of Websockets technology, which offer developers transparent use of sockets in their applications. Nevertheless, the main disadvantage of this technology is that it is very recent and has not yet been thoroughly tested. Furthermore, it works on top of the JavaScript browser engine and thus, is part of the single thread that the browser provides. Therefore, it does not support multiple threads. This can cause serious problems, because the application could become blocked while waiting for a socket response. In a system like ours, this problem should be avoided at all costs.

Due to the issues explained above, we decided to explore the option of using the Java language to communicate with the DAS server by using applets; this allows creating traditional Java applications that can run in a browser. With this option, it is possible to take advantage of all of the features that Java provides, which include the use of sockets and multiple threads within an execution layer in the browser. This enabled us to implement communication and data acquisition through TCP/IP sockets over Ethernet, based on the client/server paradigm. With this, we were able to guarantee the performance level we need for acquiring real-time data from the DAS server.

The communication protocol is proprietary and was designed specifically for communication with the DAS. The protocol features well-structured messages that are used to perform each of the different possible actions in the system, such as user authentication, real-time data requests, alarm information, historical data retrieval, etc. One of the advantages of having a proprietary protocol specifically designed for our system is that it offers good performance, as well as a good level of security.

In our implementation, communication sockets are declared within a Java object that can be accessed through the applet, which is instantiated in the Web Application (Fig. 2). Sockets can be used from JavaScript by using Java's LiveConnect technology, which enables interaction from JavaScript with calls to objects instantiated within the JVM by the applet itself. For example, this allows the application to perform the user authentication process by requesting a username and password to the user and then passing such parameters to the corresponding method in the Java object; this object in turn establishes the connection using sockets to send the authentication message.

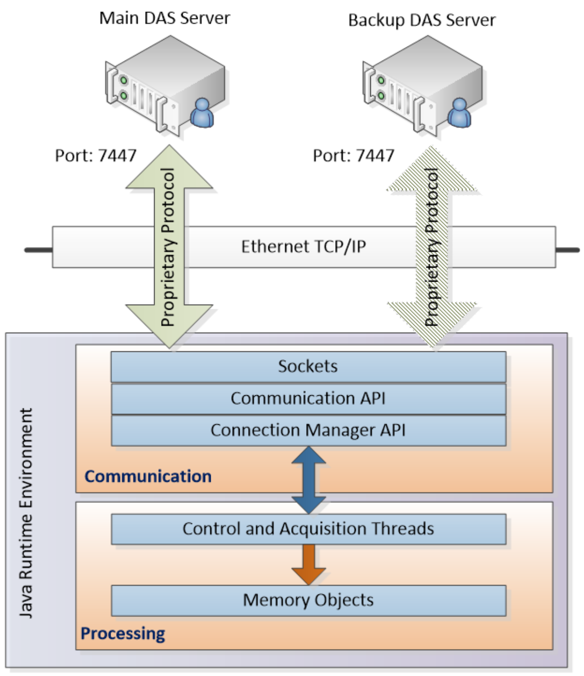

Fig. 2 Communication mechanism

The DAS can also operate in redundant mode. This means that there is a server operating as the main data server and there can be other servers working in backup mode. If the main server is down, the backup server can replace it automatically, thus avoiding any loss of information. Therefore, the communication module also has the necessary functionality to guarantee that the Web application will always be connected to one of the DAS servers. This is determined by performing periodic verifications each second, to determine which server is operating as the main data server; if the main server is down, it establishes a connection to a working server.

V. Memory Management

Memory management in real-time, Web-based applications tends to be complex, especially because most object-oriented languages contain mechanisms that dynamically create or destroy objects in memory, based on algorithms that determine when objects are no longer being used or referenced. This, in the context of real-time systems, can be problematic. Therefore, special mechanisms must be designed to properly manage objects in memory, so that application performance is not undermined. To address this problem, objects from the most important modules in our system are instantiated only once; this is done when the applet is loaded and the JVM created. In addition, these objects are static and are instantiated in a static class, which can be accessed by every object, either from Java or JavaScript. To access these objects from JavaScript, specific applet methods are invoked; these methods expose static objects stored in the JVM (Fig. 3).

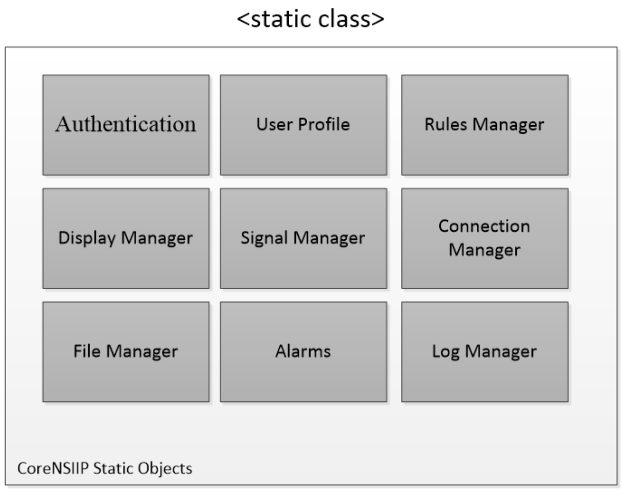

Fig. 3 Memory Objects

One issue in the interaction between JavaScript and Java is how data types are translated between both languages, when methods that return values are invoked. Native data types such as String, int, float, etc., do not cause problems. However, implementation of complex objects that include arrays can be different among browsers. To solve this problem, we use JSON objects for data exchange between JavaScript and Java. This allows us to manipulate information in a format that is common to both languages.

Another benefit that comes from implementing correct memory management mechanisms is having greater control over Garbage Collection (GC) in Java and JavaScript. Since GC cannot be disabled, different techniques can be used to delay Garbage Collection indefinitely. One of these techniques is object pooling, which consists in re-using objects instead of constantly creating and discarding new objects. This is especially important for large or complex objects.

VI. Processing

Submission By definition, the execution of a Web application is always single threaded. This means that there is only one execution thread in which all actions, such as rendering visual components, handling events or user input, managing timers, etc., are performed. This paradigm can be problematic, especially for real-time Web applications, which have very specific requirements. One of such requirements is reliability, which means that the application must be robust enough so that it can operate without interruptions during extended periods of time. Another important requirement is performance, which enables an application to react in a timely manner under different operating conditions. This aids in avoiding deadlocks or anomalies that can result in users experiencing long waiting periods or frozen screens.

The use of Java from a Web application, allows performing several operations through multiple threads that run in parallel to the main Web browser thread, thus achieving a high level of performance and reliability. Fig. 4 shows a block diagram which illustrates the threads we implemented for our application.

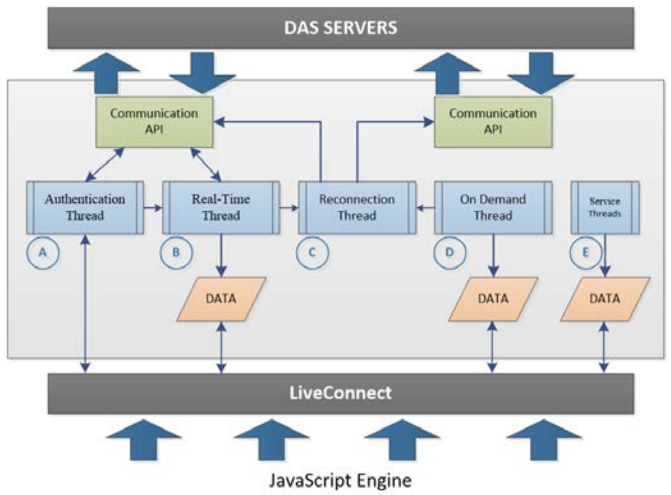

Fig. 4 Multi-Threading

A. Authentication Thread, which is created and run only once when a user first launches our application. It allows to establish a connection to the server and to send an authentication message. This thread prevents the application from becoming blocked during the authentication process. This thread also obtains certain initialization parameters, such as data point information, alarm information and server information.

B. Real-Time Acquisition Thread, which is created and run once a user has successfully authenticated and established a connection to the server. This thread operates permanently during a user session and acquires real-time data for around 8500 data points. Acquisition is performed in two cycles per second (500ms). Each cycle, data regarding software and hardware alarms, status of processes, status of acquisition subsystems and server state are obtained.

C. On-demand Data Thread, which is created and run once a user has successfully authenticated and established a connection to the server. This thread operates permanently during a user session and uses its own communication socket, which is independent from the Real-Time Acquisition Thread. The purpose of this thread is to manage user requests that involve longer processing periods, such as requests for historical data, which are often used to generate tabular or graph trend reports. These requests can take several seconds to complete due to the required data flow. If such requests were performed using the Real-Time Acquisition Thread, graphic displays would experience delays and would not be able to present the most recent information in a timely manner.

D. Reconnection Thread, which is run once, only when there is a connection problem to the server involving real-time or on-demand acquisition sockets. It also verifies that a connection is established to a server operating as the main data server. If there is no main data server online, connections are established to backup servers. Once connections have been successfully re-established, this thread exits.

E. Service Threads. These threads are created to handle requests from certain system modules that require longer processing times and that could otherwise block the execution of our Web application. For example, a request to perform an analysis on historical data for a group of data points could take several seconds or even minutes, depending on its complexity. By using a thread that is exclusively dedicated to such request, the Web browser main thread can continue to run normally, without interruption. The browser main thread would only need to monitor the corresponding service thread in order to determine when it has finished its processing, so that results can be presented to the user.

All of these threads are run asynchronously in relation to the main Web browser thread in the Java Virtual Machine, thus avoiding delays or interruptions to the browser main thread. Data which are acquired or processed by threads are stored in persistent objects in the JVM. In fact, each application module has a dedicated persistent object. These objects are accessed by JavaScript through LiveConnect calls to the Java applet, in order to obtain data which is then shown in graphic displays.

VII. Graphic User Interface

Our Graphic User Interface (GUI) was designed as a single HTML5 application. With this approach, Web-based applications do not need to refresh a whole page as the user interacts with it, similar to traditional desktop applications. To achieve this, JavaScript provides power and flexibility by enabling developers to create or modify components by accessing and manipulating the DOM (Document Object Model) in a Web page. Additionally, AJAX technology provides the capacity to obtain dynamic data from Web servers without the need to update an entire page. Fig. 5 shows a block diagram which illustrates how our Web application works.

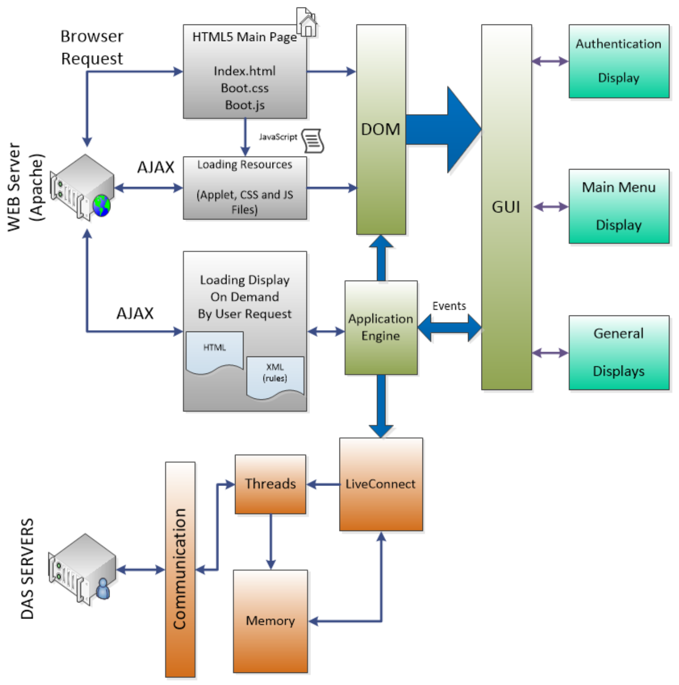

Fig. 5 User Interface block diagram

a) Our Web application consists of a single HTML index file (index.html), which includes a reference to a CSS file and a JavaScript file, which is used to load the site. Our Web application is stored in an Apache Web server, which can be accessed from any Web browser.

b) Once our Web application is loaded, a JavaScript process loads via AJAX other resources that constitute our application. These elements are: an applet which serves as an interface to Java, CSS files and JavaScript files from different modules that make up our application. The obtained resources are then inserted into the document DOM so that they become active in browser memory.

c) Once our application is loaded, an authentication screen allows users to connect to the DAS server. At this point, connection mechanisms are used to establish communication with DAS servers. It is important to note that DAS servers are different from the Web server where our Web application is hosted.

d) Once a user has authenticated successfully, the application main menu is shown. From this menu, users can open any of the graphic displays in our system. These graphic displays are implemented as HTML5 pages and can contain graphic elements based on SVG or Canvas. Graphic displays can also contain special tags which indicate whether an element has a dynamic behavior associated with it, such as showing real-time data, rendering a graph or showing information about data points or alarms. These special tags are extracted and substituted with dynamic content by the application engine, which loads the corresponding graphic display and inserts it into the DOM. It is in this post-processing phase, where graphic displays are analyzed to determine which components are to be modified dynamically.

e) The application engine is run indefinitely at a rate of four cycles per second (every 250 milliseconds). Each cycle, it obtains real-time and historical data from JVM persistent memory, for each of the data points being shown in a graphic display. These data requests are performed though LiveConnect technology by accessing applet methods, which in turn expose the requested information to the application engine via JavaScript. Additionally, the application engine dynamically modifies the content of dynamic HTML elements by manipulating the DOM and applies certain styles based on CSS files in memory. These operations are performed based on the behavior that was specified for the graphic display being shown. It is important to remember that, at this point, once a connection has been established, processing threads inside the JVM are active and acquire information from the DAS server asynchronously in relation to the main browser thread.

VIII. Conclusion

When we think about Web applications, several examples come to mind. In particular, we can think of sites that we use on a daily basis. Some of these sites are quite popular and combine some of the technologies described in this paper. However, there are very few Web-based applications that can be considered mission-critical, which operate in real-time with the level of performance and reliability required in a nuclear power plant. While designing and developing our system, we faced several major challenges, which were not easy to tackle. Despite the fact that these technologies are well-documented and widely used, it was always necessary to go one step further and discover new ways to use and integrate them. We consider that the main challenge was to achieve a correct Java applet implementation. We can say that the Java applet is the most important element in our application because it enabled us to achieve real-time communication and to implement several processing threads. The main problem with applets is also its greatest virtue: security. While the possibility of creating communication sockets, accessing disk files or accessing computer resources is very powerful, it also generates problems; therefore, it is clear that applet implementation is not simple by any means. The use of LiveConnect technology opens a wide range of possibilities for this type of applications. Inter-communication between JavaScript and Java greatly enhances processing and performance capabilities in Web applications. Currently, our application is in the final phase of development and testing. Tests have been successful since our application complies with all requirements described in this paper.

REFERENCES

[1] I. Leal, J. Suarez, E. Coronel, "A Real-Time Data Acquisition System for the Laguna Verde Nuclear Power Plant", WSEAS, July 2010, ISSN: 1109-2750.

[2] E. Coronel, "Desarrollo de un subsistema de adquisición de datos de equipos RTP, para su uso en el nuevo sistema de adquisición de datos en tiempo real de la Central Nucleoeléctrica Laguna Verde", CIINDET, pp. 4 - 5, México, Octubre 2008.

[3] E. Coronel, C. Chairez, "Subsystem of Data Acquisition Using the ModBus Protocol in Real Time of the Digital Electro-Hydraulic Control and Its Integration with the Integral System of Process Information of Laguna Verde Nuclear Power Plant", CERMA, November 2012, pp. 153 - 156, ISBN: 978-1-4673-5096-9.

[4] I. Leal, J. Suarez, "Registros históricos de tipo SCAN en memoria para un sistema de adquisición de datos en tiempo real para la Central Nucleoeléctrica de Laguna Verde", CIINDET, México, Octubre 2008.

[5] I. Fette, A. Melnikov, "The WebSocket Protocol", IETF, December 2011, ISSN: 2070-1721.

[6] HTML5 A vocabulary and associated APIs for HTML and XHTML. World Wide Web Consortium (W3C), Editor Draft's, http://www.w3.org/html/wg/drafts/html/CR/.

[7] Håkon Wium Lie & Bert Bos: Cascading Style Sheets – designing for the Web "written by the creators of CSS" (3rd edition, Addison-Wesley, 2005, ISBN 0321193121.

[8] Scalable Vector Graphics (SVG) 1.1 (Second Edition), World Wide Web Consortium (W3C), August 2011. http://www.w3.org/TR/SVG/

[9] D. GoodMan, "JavaScript Bible 3rd Edition", Chapter 38, ISBN: 0764531883